



TECHNISCHE UNIVERSITÄT
CHEMNITZ

Department of Computer Science

Data Management System Group

Master Thesis

Design and Implementation of a Web-Based Software for the
OPC Unified Architecture Integrated into a Semantic Ques-
tion Answering in the Domain of Smart Factory

Orcun Oruc

Chemnitz, 18 March 2019

Examiner: Dr. Frank Seifert

Supervisor: M. Sc. Adrian Singer

Orcun Oruc, Design and Implementation of a Web-Application for OPC UA Protocol Integrated into a Semantic Question Answering, Master Thesis, Department of Computer Science Chemnitz University of Technology, 18 March 2019

Sperrvermerk

Diese Master Thesis enthält vertrauliche Daten der Fraunhofer IWU Institute. Eine Veröffentlichung dieser Arbeit, auch auszugsweise, ist ohne ausdrückliche Genehmigung der Fraunhofer IWU Institute nicht zulässig. Diese Arbeit darf nur den Korrektoren und dem Prüfungsausschuss zugänglich gemacht werden.

Abstract

Manufacturing technologies in the manufacturing systems have been shaped over the past few years. Such changes in information technology, data analysis and manufacturing systems has enabled data collection and this concept incarnated as smart factory in industrial plants.

Nowadays, a factory generates more obscured data from manufacturing process and industrial communication rarifies the connectivity between manufacturing devices. Those issues on manufacturing systems have changed their demands in terms of smart factories that aim to boost performance and productivity in manufacturing. In this context, machine-to-machine protocols have been evolved in helping to shape requirements relevant to production systems. Service-oriented architecture and compatibility to high-level client-server communication put forward the OPC Unified Architecture. OPC Unified Architecture (OPC UA) is a de-facto protocol in the usage of communication at the industrial scale of smart factories, sensor networks, and manufacturing systems. The OPC Unified Architecture supports eliminating the dependency of factory-level communication and creating a vendor-independency in smart factories.

One of the main problems affecting the smart factory is non-uniform and lack of standardization, which could examine a factory system without knowing the underlying structure. The latter is an assistant question answering that enables natural input to answer operators. Even though a smart factory generates a massive amount of data

ABSTRACT

through industrial processes, technical personnel or operators cannot easily interpret the linked data created by different sources. A semantic question answering can reply questions of the operator and experts that posed in a natural language. Those questions can consist either streaming data or static data regarding industrial communication.

To tackle those two problems as a whole, this work proposes an architecture design as well as a robust implementation in the web-based platform, which chiefly focuses ease of integration and usage. In essence, the goal of this thesis is to create an operator assistant web-based software. To achieve this goal, the research will orchestrate a particular machine-to-machine protocol and human to machine approach that serves as a web-based software.

Notably, the thesis will examine the integrated web applications relevant to OPC Unified Architecture and assess the applicability of the semantic question answering aspect of the natural language understanding.

The practical implementation of this research follows a staged approach; we then examine architectural requirements and viability of OPC Unified Architecture to the web environment; besides, we will provide results about the viability of the semantic question answering system in terms of human-machine interaction.

Consequently, this thesis exemplifies a practical implementation to evaluate the operator assistant web-based software in smart factories. This research will be an innovative tool with significant findings for the future researches in the sense of human-machine application integration into the OPC Unified Architecture.

ACKNOWLEDGEMENT

Acknowledgments

This research was supported by the Fraunhofer IWU Institute. I especially thank my supervisor and colleague M.Sc. Adrian Singer. Without his guidance and help, this thesis would not have been possible. I would like to express the deepest appreciation to my university supervisor Dr. Frank Seifert who provided insight and expertise that greatly assisted the research. I thank Diplom Inf. Ken Wenzel for assistance with LinkedFactory and Enilink Systems[1] and for his comments that remarkably improved the manuscript. Finally, I am deeply grateful to my family who supported whatever it costs through all my life.

Table of Contents

List of Figures	xi
List of Tables	xiii
List of Listings	xv
List of Abbreviations	xvii
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	2
1.3 Objectives and Scope	3
1.4 Organization and Contributions	6
2 State of the Art	8
2.1 Background of OPC Unified Architecture in Web Environment	8
2.2 Background of the Question Answering	11
2.3 Linked Data Collection for Heterogeneous Data Source	16
2.3.1 OPC Unified Architecture Information Model Serialization	16
2.3.2 Linked Data Collection from Streaming Data.....	17
2.4 Chapter Discussion	19
3 Industrial Communication in Smart Factories	22
3.1 Human Machine Interaction and Smart Operators in Smart Factories	23
3.2 OPC Unified Architecture for Web Applications	24
3.2.1 OPC UA Service Sets	24
3.2.2 OPC UA Address Space Model	25
3.2.3 OPC UA Information Model	27
3.2.4 OPC UA Discovery Service	29
3.2.5 OPC UA Subscription Service	30
3.3 Chapter Discussion	32
4 Theory of the Semantic Question Answering over Linked Data	33
4.1 Semantic Web Technologies	35
4.2 Data Preparation for Heterogeneous Data Source	39
4.3 Natural Language Processing in Question Answering	40

TABLE OF CONTENTS

4.4 Chapter Discussion.....	52
5 Practical Implementation	53
5.1 The RESTful Architectural Design in the Assistance Web-Based Software	53
5.2 Front-End Development.....	57
5.2.1 Overview	57
5.2.2 Implementation of Front-End Development.....	59
5.3 Back-End Development	60
5.3.1 Overview	60
5.3.2 Authentication and Discovery of the Web-Based Software	63
5.3.3 Connection Establishment of OPC UA	66
5.3.4 Data Navigation and Polling through OPC UA in the Web-Based Software	67
5.3.5 Implementation of the Semantic Question Answering.....	69
6 Experimental Results	74
6.1 Test Methods and Environment.....	74
6.2 Question Answering Data Sets and Evaluation Metrics.....	76
6.3 Results	77
7 Discussion	86
7.1 Introduction.....	Error! Bookmark not defined.
7.2 Assessment of Research Questions and Hypotheses	89
8 Conclusion	90
8.1 Summary.....	90
8.2 Future Works.....	91
Bibliography	93
Appendix A The Semantic Question Answering	99
A.1 Manually Developed Test Questions – Precision and Recall	99
A.2 Natural Language Understanding Libraries	102
A.3 KVIN Service Sample Query.....	104
A.4 KVIN Service Result of Appendix A.4 with a Key-Value Pair.....	104
A.5 Serialized Streaming Data into Linked Data	105
A.6 KVIN Streaming Data SPARQL Service.....	106
A.7 eniLINK Prefixes.....	106

TABLE OF CONTENTS

A.8 Deep Parsing-Shallow Parsing	107
A.9 Generated Answer from OPC UA Information Modeling.....	108
A.10Transformed RDF/XML Data into Turtle RDF.....	109
A.11Transformed XML Data into RDF/XML.....	110
A.12 Question Classification.....	111

Appendix B Web Application 113

B.1 Theoretical Background of Architectural Thinking of the Web-Based Software	113
B.2 CoffeeScript Sample.....	117
B.3 JavaScript Counterpart of the CoffeeScript Sample	118
B.4 Script Languages for User Interface Development.....	118
B.5 Dynamic Server quasi OPC-UA Server.....	120
B.6 OPC Unified Architecture Information Model Serialization Algorithm.....	122
B.7 Backend Framework Comparison	123
B.8 Frontend Framework Comparison	124
B.9 Load Balancer & Reverse Proxy Configuration	125
B.10HTTP Headers for Evaluation	126
B.11HTTP Requests without Load Balancer	126
B.12HTTP Requests with Load Balancer	127
B.13Question Answering HTTP Request	127

Appendix C Literature Review 129

C.1 Question Answering Summary.....	129
-------------------------------------	-----

Glossary CXXXI

List of Figures

Figure 1.1: Organization of the Thesis	7
Figure 3.1: OPC UA Address Space [42] [41]	26
Figure 3.2: OPC UA Information Model [45]	28
Figure 3.3: Subscription and Monitoring Item Services [41].....	32
Figure 4.1: Maximum Likelihood Estimation [46].....	43
Figure 4.2: Perplexity formula of a language modeling [46].....	44
Figure 4.3: Named-Entity Recognition by Stanford CoreNLP	48
Figure 4.4: Person and Organization assignment by AllenNLPError! Bookmark not defined.	
Figure 4.5: Jaccard Similarity Formula [56]	49
Figure 4.6: Jaro Formula [57]	49
Figure 4.7: Levenshtein Formula [57].....	50
Figure 4.8: Wu Palmer Formula [59]	50
Figure 5.6: RESTful Architectural Design of the Web-Based Software (discovery server and load balancing & reverse proxy should be added).....	54
Figure 5.1: Authentication System in the Practical Implementation	65
Figure 5.2: The Flowchart of the Semantic Question Answering.....	70
Figure 5.3: Query Formulation Algorithm: The source code can be found under the link https://github.com/zointblackbriar/QuestionAnswering	72
Figure 5.4: Dependency parser.....	73
Figure 5.5: Named-entity recognition	73
Figure 6.1: Single REST Request Response Time.....	79
Figure 6.2: Multiple REST Request Response Time	80
Figure 6.3: Total Amount of REST Requests	81
Figure 6.4: Single REST Throughput.....	82
Figure 6.5: Multiple REST Throughput.....	83
Figure 0.1: Enilink Sample SPARQL Query	104
Figure 0.2: A result from a continuous data.....	104
Figure 0.3: KVIN Service Relationships with Components	106
Figure 0.4: Constituent parse tree for factoid questions.....	107
Figure 0.5: The Dynamic Server Publish/Subscribe Model.....	121
Figure 0.6: Extraction Algorithm of OPC UA Address Space. The source code can be found: https://github.com/zointblackbriar/QuestionAnswering	123

List of Tables

Table 3.1: Discovery Service Pros and Cons.....	30
Table 6.1: Functionality Test Results	78
Table 6.2: Evaluation parameters of the Semantic Question Answering.....	84
Table 6.3: Total answers from Semantic Question Answering.....	85
Table 0.1: Precision and Recall of Answers	101
Table 0.2: NLP Toolkits Advantages and Drawbacks	103
Table 0.3: The Question Classification of Li&Roth and Wh-Question Taxonomy ..	111
Table 0.4: Types of Middleware Publish/Subscribe	122
Table 0.5: Backend Development Framework [64].....	124
Table 0.6: Script Languages	124
Table 0.7: Question Answering Summary.....	129

List of Listings

Listing 4.1: Sample SPARQL against a generated local source	39
Listing 4.2: Wu Palmer Sample Calculation[58]	51
Listing 4.3: Leacock-Chodorow Sample Calculation[58].....	51
Listing 5.1: HTTP Get Request for token-based authentication	55
Listing 5.2: Http Get Request [7] [6]	55
Listing 5.3: Http Post Request [7] [6].....	56
Listing 5.4: Question Answering Static Message HTTP	56
Listing 5.5: Question Answering Dynamic Message HTTP.....	56
Listing 5.6: HTTP Get Request for Monitoring Node	56
Listing 0.1: Generated RDF from Real-Time Data Source.....	105
Listing 0.2: Enilink Sample Prefixes	106
Listing 0.3: An answer from generated OPC UA Semantic Data.....	108
Listing 0.4: Preview of Generated Semantic Data from an OPC UA Server.....	109
Listing 0.5: Transformed XML Data into RDF/XML.....	110
Listing 0.6: A sample from Coffeescript	117
Listing 0.7: Counterpart of sample CoffeeScript in Figure 1.1.....	118
Listing 0.8: Sample configuration of a load balancer	125
Listing 0.9: HTTP Header of Load Test.....	126
Listing 0.10: Multiple Requests without Load Balancing.....	126
Listing 0.11: Multiple Request with Load Balancing	127
Listing 0.12: Question Answering HTTP Request.....	127

List of Abbreviations

NLP	Natural Language Processing
HMI	Human Machine Interface
REST	Representational State Transfer
JSON	JavaScript Object Notation
JWT	JSON Web Token
OPC UA	Open Platform Communication Unified Architecture
Fraunhofer	Fraunhofer Institute for Machine Tools and Forming
IWU	Technology
VDMA	Mechanical Engineering Industry Association
MVP	Minimum Viable Product
RFC	Request For Comments
RDF	Resource Description Framework
SOA	Service Oriented Architecture
OLE	Object Linking and Embedding
SDK	Software Development Kit
CLR	Common Language Runtime
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
W3C	World Wide Web Consortium

LIST OF ABBREVIATIONS

XML	Extensible Mark-up Language
DOM	Document Object Model
MVC	Model-View-Controller Pattern
SDK	Software Development Kit
HTTP	Hypertext Transfer Protocol
QA	Question Answering

1 Introduction

1.1 Motivation

"The most obvious characteristic of science is its application: the fact that, as a consequence of science, one has a power to do things. And the effect this power has had need hardly be mentioned. The whole industrial revolution would almost have been impossible without the development of science."

- Richard P. Feynman

Today, human operators and experts have too many generated data, and complex protocol structures that can be complex depends on vendors of devices and facilities in an industrial plant. Advanced manufacturing evolved with the technology called 'Industry 4.0' that enables unstructured linked data through the devices where dispatch messages with industrial communication protocols. Industrial plants and their automation processes have been digitized and equipped with sensors and actuators. Then an infancy term came to exist so-called 'Smart Factory'. Smart factories apply operations that should be able to run without much human operator's intervention. Connectedness among sensors, machines, and systems created complex communication infrastructure, beyond that the generated data by smart factories became more complicated to understand and interpret for human operators.

More specifically, information gathering through complex, unstructured data and communication architectures in a smart factory have a critical role for human operators and experts. If we ignore the problems, gathering information would be time-consuming and tedious while wading through a large number of semantic documents with semantic queries can also increase training and operation time. The process of collecting technical information may aggravate the mobility of human operators and the productivity on an assembly line or a manufacturing device.

1 INTRODUCTION

We want to design and implement web-based software to solve the problems above. This web-based application should be high performance, ease of use, robust, secure and scalable according to the needs of a smart factory. Technology considerations of the operator assistance web-based software may orchestrate with the architectural design to alleviate the problems mentioned above for human operators and experts. More importantly, it should respond to the requirements of a smart factory aspect of analytics of data.

In order to design and implement the web-based software, we should able to traverse vendor-specific platform-independent data of OPC Unified Architecture or interlinked heterogeneous data source with a user interface. Tackling the problem in the smart factory domain, we aimed to design and implementation of an operator assistance web-based software that can answer the limited type of questions related to pieces of equipment of manufacturing and traverse data among various industrial devices through OPC Unified Architecture with limited knowledge of underlying structure.

Lastly, Fraunhofer IWU continuously researches connected sensors, and actuators' data brings us a need for further understanding of the meaning of data. The semantic question answering can interpret the meaning of data, a heterogeneous data source should handle the produced data by smart factories and by the information model of OPC Unified architecture. The operator assistance web-based software orchestrates an industrial automation protocol and Natural Language Understanding, which can increase the ability of adaptive automation between human operators and manufacturing devices.

1.2 Problem Statement

We want linked data of manufacturing devices can be seamlessly comprehensible by experts, with limited information about complex technical architecture, the experts should also can aware the internal process of the industrial devices utilizing natural language queries.

The problem that we faced is a necessity of an aggregated information retrieval-industrial communication suite at a smart factory of Fraunhofer IWU by utilizing the company-specific data. Current researches do not tackle the problem as a whole in industrial manufacturing. The issue that we encountered can influence human operators or factory workers spend a considerable amount of time on operating machines by using smart

devices. When a new technical personal attends in a smart factory, the web-based software can reduce the training cost by giving general information about a particular manufacturing process of a facility.

However, the lack of technical information about industrial automation the underlying about industrial automation is a significant problem for experts who work in smart factories. Human-centered assistance application design can show error logs where the error occurred in manufacturing devices. A control module can poll instantly servers that are up and running by utilizing OPC Unified Architecture. An operator can observe simulated data without installing any software from anywhere. Moreover, an expert can plan the process while look at interlinked data in the heterogeneous data source that consist of streaming data or static data.

We cover this research in the context of industrial automation at different facilities in the same smart factory of Fraunhofer IWU. After describing the necessary parts of OPC Unified Architecture and features of the question answering, we need to implement the operator assistance web-based software.

The web-based software implies that a design decision of the web-based software can change the robustness and efficiency of the web application. Not only we can think of a web-based software deploying for a single manufacturing device of a smart factory, but also we consider that a large number of users can connect to this system. Therefore, a robust and scalable design has an essential role in deploying the overall system. In case we obtain a solution, there will be an innovative solution that can implement into a smart factory, which supports increasing the efficiency outcome at the manufacturing scale.

We will use a top-down approach to design and implement the web-based application with state-of-the-art web technologies by taking into consideration of architectural design thinking. More specifically, our methods rely on frameworks of back-end and front-end development that integrated into various libraries of natural language understanding.

1.3 Objectives and Scope

The goal of this thesis is to create an architectural view that serves as an operator assistant web-based software. Particularly, the operator assistant web-based software brings

1 INTRODUCTION

into together subdisciplinary concepts of the computer science so-called question answering system and OPC Unified Architecture machine-to-machine communication protocol by assessing the viability of web technologies and suitability of natural language understanding concept.

This thesis reviews the past literature to indicate the gap in knowledge and possible limitation while linking the internal data of a particular service that resides in OPC UA and orchestrating the semantic question answering. As a part of this work, we need to clarify the current state of research on scientific publications regarding important inquiries that guide clearly in a literature review. The study has two-fold research methods, which are theoretical review and practical implementation.

This research scope limited with the smart factory that is relevant to manufacturing technologies. The part of OPC Unified Architecture connectivity will be restricted with definitive services that would comply with our research questions. Due to the limitation of data scope, the semantic question answering system will answer questions regarding industrial automation.

This thesis also reviews past literature to indicate the gap in knowledge and possible limitation while creating a heterogeneous data source and linking of an OPC UA Information Model with the data source. We will have some limitations while conducting the research, which are:

The first limitation is that we will implement a module that allows querying, writing and monitoring for current data assessing the optimal web software architecture at a basic level. We are not aiming to research historical data access and alarms.

The second limitation is that data sources can be streaming key-value mapped data or generated data from OPC UA Information Model, but the sample size is relatively small for ontology-based systems. The data source does not contain documents, plain text or open linked data. A couple of data source has been created for the sake of reaching research goals. As of being, a persistent data storage, so-called the graph databases, will not be considered in this work.

The third limitation is that the manually created test questions are mandatory to evaluate for the semantic question answering.

The fourth limitation is deployment technologies such as containerization, or continuous integration technologies will be out of scope. The scope also does not include cloud-

based and serverless architectural pattern when it comes to the design of web-based software.

The scope does not cover creating a full-fledged application that browses and modify every possible data types in OPC Unified Architecture; rather, our scope covers designing of the operator assistant web-based application at the architectural level and the quantitative and qualitative evaluation. Likewise, the semantic question answering component does not address paragraph-based input and unlimited data domain with unlimited types of questions.

Last but not least, we have defined research questions to consider possible implications, and we try to answer at the end of the research. To reach the primary goal, I have set up research questions and hypotheses

These research questions and hypothesis as below:

My fundamental research question is:

- How can I design and implement an operator assistant web-based software with a web architecture that consists of semantic question answering and OPC Unified industrial communication technology in the domain of smart factory?

Important sub-questions that relevant to the part of operator assistant web-based software:

RQ 1) Are the components of OPC Unified Architecture well enough to perform for a factory-wide deployment concerning industrial communication?

RQ 2) What would be the optimal web architecture aspect of robustness, security, ease of use and high performance to implement the operator assistant web-based application in a smart factory?

RQ 3) Can a semantic question answering utilize restricted domain heterogeneous linked data source (e.g., OPC UA based data, streaming data, static data) and how well perform a question answering in terms of our scope?

RQ 4) Can we scale our approach to other smart factories or plants?

My hypotheses are the following:

- H 1) The modules of proposed architecture will not affect each other in the context of performance and functionality.
- H 2) The proposed system will provide correct and rapid results to human operators.
- H 3) Existing methods and design principles in the previous researches can be applied to the operator assistant web-based software.

1.4 Organization of the Study and Contributions

This thesis contributes to the research circle in the following ways:

- We introduce an innovative web-based software by demonstrating a detailed architecture for realizing an assistant software that is robust, secure, performative, and ease of use. The architecture integrates web application designing, web architecture, OPC Unified Architecture industrial communication, and Natural Language Understanding.
- We propose a synthesized idea through transdisciplinary areas of computer science so-called web science, communication network and artificial intelligence to make an assistant software for human operators in a smart factory.
- We propose a human-machine interaction tool that can answer to factoid questions from various linked data source data sources such as time-series data and semantic OPC UA data.
- We propose a heuristic-based syntactic analysis to solve the question answering challenges in a restricted domain question answering.
- We introduce an industrial communication tool integrated into a semantic question answering to aware of context information in a smart factory.
- We evaluate the architecture with qualitative researches through performance and functional experiments and provide empirical results to the readers. Also, we evaluate the question answering part with manually generated questions.

1.4 ORGANIZATION OF THE STUDY AND CONTRIBUTIONS

This research organized as follows;

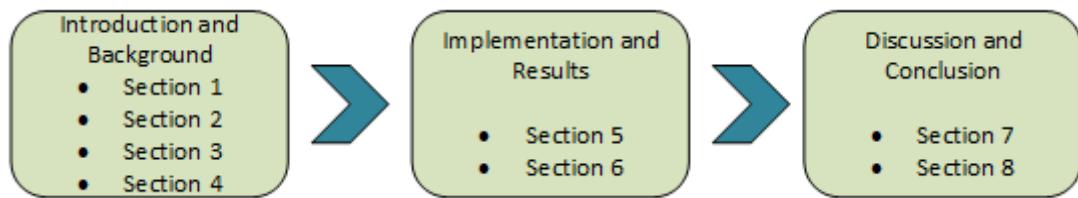


Figure 1.1: Organization of the Thesis

Section 1 presents motivation, problem definition, scope, and related works. Section 2 conducts a literature reviews regarding OPC UA, Question Answering and Linked Data Collection that are examined separately.

Section 3 introduces industrial communication in smart factories in terms of theoretical background of operator assistant and human-machine interaction. Section 4 explains the theory of the Semantic Question Answering in the context with natural language understanding. In Section 5, we will elaborate on the details of practical implementation with architectural details.

Test preparation and results will be in Section 6, and we will list the results under the section. Before we reach a conclusion, we will comment on our results and express our results are meaning to readers in Section 7. Finally, the conclusion part will define our findings and future improvements in Section 8.

2 State of the Art

We will discuss the state-of-the-art research by splitting up Section 2.1, Section 2.2, and Section 2.3. Each section will give information towards sub-questions that have mentioned previously. In the end, we will summarize the state-of-the-art researches by discussing essential points that could steer with the rest of the thesis.

2.1 Background of OPC Unified Architecture in Web Environment

The following statements give brief information about the historical development process of OPC Unified Architecture. On the path of development, OLE OPC was formerly known as OLE for Process Control has been widely dispersed at the industrial scale. The term Object Linking and Embedding for Process Control traced back to initial development that was founded by Microsoft in order to communicate objects with Component Object Model (COM). Object Linking & Embedding was a collection of operating system services that allowed anyone to include component objects in an application or allow one to package component objects for use in other applications[2].

The principle of OLE is to develop modular applications that refer to a ground-breaking step for loose coupling in object-oriented programming. The use of object-oriented techniques encouraged reusability and maintainability to industrial communication. OLE has utilized component objects that can have relations with other objects. The objects that belong to the OLE Model could support interfaces to deploy an abstraction layer for other objects. The ultimate drawback of OLE components that do not support inheritance to increase the integrity of run-time development [2]. This issue led to another issue, which multiple vendors can create multiple types of objects and releases, and yet, OPC Unified Architecture development demanded to solve multiple releases issue through data integrity in the communication stack.

The second development was the OPC Classic that shaped an architecture using the client-server model for information exchange. The advantage of the approach in OPC Classic can employ the definition of different APIs for different specialized needs without a network protocol definition for inter-process communication [3]. Unfortunately, OPC Classic suffers how to enhance security in industrial networks when OPC Classic was used. Along with the OPC Classic, the first data access modeling has been developed for

2.1 BACKGROUND OF OPC UNIFIED ARCHITECTURE IN WEB ENVIRONMENT

reading, writing and monitoring the value changes. When a user requires inserting a simulative model with metadata, OPC Classic supported the simulative model in a limited way. Taking into consideration the limitations of OPC Classic, OPC UA came out into industrial communication.

OPC Unified Architecture (OPC UA) was developed by considering the drawbacks of the traditional OPC Classic Platform, which is a platform-vendor independent and service-oriented architecture. The primary motivation for migrating from OPC Classic is that its message protocol based on the Component Object Model, so OPC UA should have supported multiple communication protocols and operating systems [4].

The OPC Classic did not aim to connect end-user devices to the underlying protocol. To remedy this drawback, OPC Unified Architecture, integrates all the functionality of the individual OPC Classic into an extensible framework turned out to be a de facto standard and released in 2008 [5]. After obtaining a design regarding loosely coupled, coarse-grained, and service, OPC Unified became capable of integrating into web applications.

Few researchers have addressed the issue of the viability of OPC Unified Architecture on a web platform. [Cavalieri, Salafia & Scroppi 2018] [6] make an effort to enhance interoperability with web technologies to comply with web environment using Representational State Transfer mechanism. After publishing an article [6], they proposed research for end users who do not know the technical background about OPC UA Protocol and one can use a web application that provides a token-based authentication[7]. The research studies referenced as [6][7] offer a new concept for loose-coupling architecture at back-end development and advanced subscription system and asynchronous message broker protocol for MQTT, AMQP, and SignalR.

Furthermore, the authors listed and implemented the most crucial service elements such as SecureAccess, GetDataSources, ReadInfo, WriteInfo and Subscription through assuring grant access to the services. They have implemented an application for Publish/Subscribe architecture of OPC Unified architecture with REST calls. Besides, the authors have designed the whole system in a unified backend architecture without a front-end architecture. The drawback of these papers is that it does not implement a front-end application to scale in a web architecture in the context of user interaction and performance. In addition, the authors have mentioned the discovery service of OPC Unified Architecture, but they did not express how to integrate into web application architecture or how it could be beneficial for a web application.

The authors justified the algorithm decision as middleware, user request module, and OPC Client Module. The studied algorithm reproducible but not easy to configure for testing [8]; nevertheless, it can give us insight and design decision. The implementation and research have conceptual differences about REST API definitions, Publish/Subscribe implementation, and serialization of data. However, [Cavaliaeri, Salafia & Scroppi 2017] emphasized that it is a novel approach based on a Publish/Subscribe pattern and stated all other solutions that relate to RESTful API integration into OPC UA requires handling the communication stack of OPC UA [7]. [Paronen 2015] stated that the author examined the requirements for the generic client and concerned with the selection of technologies as back-end development and front-end development. The author defined the core problems that are mixing the techniques in a monolithic application, stateless mapping API onto stateful OPC UA Sessions, incapable of supporting multiple service instance, and a performance bottleneck in the web client aspect of round-trip times between the client and the service [9]. Accordingly, the author emphasized that industrial plants generate vast amounts of data, which is a need for semantic understanding data and decision-making software [9]. The author implemented the features of reading over Historical Data, browsing in the Address Space, Subscriptions and calling Methods through the service layer and presentation layer.

[Grüner, Pfrommer & Palm 2015] introduced a concept relevant to RESTful integrability of OPC UA [10]. The main advantage of this paper has quantitative results to show performance through transport layer protocols such as TCP and UDP, and they have given a clear comparison among various types of machines such as Raspberry Pi, X86 PC and WAGO Device [10]. At the protocol level of OPC UA, they have introduced a good concept with a stateless and stateful process. Although this approach is interesting, it suffers from practicability of service sets with web platforms. [Grüner, Pfrommer & Palm 2016] published another paper that has significant results about the communication between REST and OPC UA by referring to reduced communication overhead, caching layers, stable service interfaces [11]. Unlike [10], they have mentioned the importance of load balancing the caching ability of REST on the Network Level [11]. In any case, these two types of research have not targeted to implement a web-application on the application layer, but evaluations could be valuable in terms of session initialization in OPC UA and outcome over the roundtrip time of REST Protocol on OPC UA Sessions.

[Shiekofer, Scholz & Weyrich 2018] attempted to map OPC UA Protocol onto Representational State Transfer System by listing the features that they need. The authors empha-

sized the main problems such as HTTP Mapping, Sessionless Invoke and Browser Support[12]. It is generally accepted that their problem sets, but full-scale integration architectures, an architectural overview of the application, and details of implementation have not been addressed.

2.2 Background of the Question Answering

Principally, a question answering system is a system to answer a question by human interaction concerning information retrieval and natural language processing theories. Open-Domain Question Answering identifies the question can be asked to a general type of data sources such as DBpedia, Freebase, Wikidata and so on. Not only a specific domain can be queried, but also a user may request in any question so as to get an answer from a data source. Closed-Domain Question Answering allows that a user may ask a question against a restricted type of data source that has defined in which a commercial domain resides. A user cannot ask all kind of questions so as to get an answer from a restricted data source. Closed-Domain is a broader term than restricted domain; hence, we will use restricted-domain question answering or more specifically semantic question answering in the rest of the research. A semantic question answering exploits linked data or semantic triples, which could represent ranging from linked open data cloud to a restricted domain. Linked open data cloud might be associated with any topic such as geography or social networking, and it has been made by multiple institutions, department and research facilities all over the world. Whereas, our domain type is restricted with a restricted factory domain so that we initially will focus on researches about the characteristics and features of a semantic question answering at this chapter. Then we ought to examine algorithms and application that has specified in domain-specific question answering.

Regarding the Semantic Question Answering, researches mostly focus on algorithms on how to transform from a natural language query to SPARQL Protocol and Resource Description Framework (RDF). A semantic question answering can use different types of dataset range from structured data to streaming data. Primary data sources are plain-text documents, open data cloud (Wikipedia, DBpedia), time series value, and linked triple data.

Few researchers have addressed the problem of restricted-domain question answering. [Molla, Gonzalez Et al. 2007] overview the main characteristic of a question answering

2 STATE OF THE ART

in restricted domains is the integration of domain-specific information either developed for question answering or disclosed for other purposes[13]. The authors define the main characteristics of the question answering system over limited domains as below [13];

- It should be circumscribed
- It should be complex
- It should be practical

Circumscription motivates a user should know what kind of questions are available to the domain. [Molla, Gonzalez Et al. 2007] assumed that the more restricted domain is, the more likely obtaining data from comprehensive databases [13]. Developing a system in a complex domain, it might become difficult to manage all context of data. [Molla, Gonzalez Et al. 2007] emphasized that users should be aware of the level of detail expected from the answers and frequently searched questions in a practical manner [13].

The authors have compared between open-domain and restricted-domain question answering by figuring out key points. According to their paper, they have defined three clear-cut subjects, which are [13]:

- The size of data
- Domain Context
- Use of Domain-Specific Resources

[Molla, Vicedo 2007] figured out the main issue that they defined the restricted domain question answering may not use the ontologies of the open domain because it has too fine-grained structure [14]. The authors emphasized that developing a system in a specific domain could be time-consuming; therefore, one should consider porting a framework from other domains [14]. [Tirpude, Alvi 2015] presented a closed-domain question answering for law documents, which employs question processing module, document processing module, and answer processing module respectively [15]. As being a plain-text document-oriented question answering, the authors developed an algorithm in answering questions for plain-text documents by scoring the created answers. The practical implementation has been carried out clearly; hence the authors reached some results such as F1-Score = 0.62, precision = 0.92, and recall = 0.62 within 100 questions overall. Example of questions has been constructed mostly factoid questions that means constrained with “Wh-Questions”. [Chung Et al. 2004] has been proposed a restricted

domain question answering that works with weather forecast data. They have used a named entity tagger, and dependency parser was used to analyze the question precisely [16]. Although their practical system transforms natural language queries into the relational data query known as SQL, [Chung Et al. 2004] mapped the particular keywords onto the column name of the relational database. Answers were generated with a rule-based method which each query frame has an answer generation pattern for a frame [16]. [Chung Et al. 2004] has designed a paper that did not elaborate the algorithm, but we can take into consideration the precision and recall values which are 90.9 % and 75.0% respectively [16].

[Diefenbach Et al. 2017] published a survey paper the techniques about Knowledge Base Question answering with technical challenges, possible datasets and required common methods in knowledge based open domain question answering. They have listed evaluation methods such as the recall and the precision and pointed out syntactic features, phrase mapping on <subject, object>, property mapping onto <predicate>, dependency between the different phrases [17]. The aforementioned methods mostly founded the stage of question analysis. According to this survey, constituency parser on phrase structure, dependency grammar on relationship among the parsers might be used. N-gram language modelling and named-entity recognition can identify contiguous spans of tokens [17]. SPARQL Query constructions have been made either using templates or semantic parsing regarding syntactic parsing of noun phrases and verb phrases. To detect a set of similar phrases for a property, they emphasized string or semantic similarity. Lastly, in their conclusion, they have stated that examination of existing techniques individually is nearly impossible [17].

[Nguyen, Kosseim 2004] focused on the problem of precision performance in a restricted question answering [18]. The authors stated that the TREC or regarding open-domain question answering test datasets are less helpful for evaluating a restricted domain question answering. They criticized that lexical and semantic techniques such as WordNet similarity analyses may not apply well in the context of a restricted domain question answering [18]. The authors designed a term score system that trained with the predefined particular keywords to increase the precision of the question answering. The data source of this restricted domain question answering is a manually collected document set. The authors created a system called Okapi Formulation that has reached with 60 questions to 53.8% accuracy rate under a particular document set [18].

We have introduced characteristics and features so far. Currently, we will examine the algorithms regardless of being open-domain or restricted-domain.

[Dwivedi, Singh 2013] briefly survey the significant characteristics and algorithm types of QA Systems. They have defined the approaches as a linguistic approach, statistical approach, surface pattern based, and template based. The authors compared the linguistic, statistical, and pattern-based approaches by semantic understanding (Deep or Shallow Parsing), heterogeneous data handling, reliability, scalability, evaluation technique (manually developed, TREC¹, CLECT, NTIRC test set and so on) and application area (open domain, restricted-domain etc.) [19] in the discussion of the paper.

[Tatu Et al. 2016] proposed an article that described a semantic question-answering engine for merged structured and unstructured datasets [20]. Even though their proposal may process on generated semantic triples from a plain-text document on the biomedical domain, triples were created labeled such as “<lymterms: text> won </lymterms: text>”. Another advantage of the paper of [Tatu Et al. 2016] is calculating semantic closure between lexical chains by implementing a hybrid identifier with the part-of-speech, lemma, parsing path to Wh-word, and named-entity recognition [20]. The authors followed a heuristic approach with answer ranking after making a query formulation, and they tested over 232, 585 n-triples with the mean reciprocal rank formula (MRR) [20].

[Celikyilmaz 2006] proposed a Bayesian Model method in different fields of natural language processing to help extract information from unstructured text. A probabilistic method that each topic-word in a document assigned to the 50 fine-grained named entity types were used [21]. The Latent Dirichlet Allocation has been used to search for a probabilistic match given topic and word in terms of word-topic position. One major drawback about the research is lack of evaluation of the algorithm.

[Unget Et al. 2012] defined a problem that most of the questions answering systems translate the question into a triple to match RDF data directly in open-domain question answering [22]. The authors proposed a solution to remedy the problem by creating a SPARQL template that provides a straight match into the internal structure of a question. They applied similarity metrics and search heuristics that consist of named-entity recognition, semantic representation by parsing, and POS Tagger [22]. The main advantage of the paper is that the system tried to detect properties of triples employing string similarity algorithms for entities such as Levenshtein and substring regex finder [22]. The main disadvantage of the algorithm is that the system does not care about the adjoining subtrees among entities except for the interchanged relationship between verb and nouns.

¹ <https://trec.nist.gov/data.html>

[Palaniappan, Sridevi & Subburaj 2018] focused on a question answering system by generating a template-question and semantic similarities of inputs in the e-learning domain. This work aims at a different type of questions with different patterns that have to be matched with the ontology tree structure in a document-oriented closed-domain [23]. Their architecture consists of tree tagger, WordNet similar matcher, ontology-query mapper, and a POS Tagger. There is a tree tagger parser to identify question patterns such as “give”, “define” or “what”, instead a question classification does. Synonyms of the noun/verb/adverb and adjective were checked with WordNet to map onto ontologies [23].

[Ferre 2012] has published one of the detailed research that expresses common pitfalls of natural language processing, essential points while consolidating SPARQL query language and morphological definitions [24]. SQUALL is a solution for querying and updating RDF graphs by exploiting a controlled natural language which restricts grammar structures of a sentence in order to diminish the complexities aspect of morphological structures the given input [24]. It has grouped all substantial features of a morphological language and pointed out what type of elements in a natural language harnesses with regarding priorities and orders. The main contribution of [Ferre 2012] is categorizing ambiguities of natural languages and advantages of using a controlled natural language by sketching a translation from their intermediary language to linked data triples to gain more accuracy with their system [24]. [Luz, Finger 2018] aimed to transform natural language sentence into a query of SPARQL. They have used so-called dataset Geo880 that contains geographical questions and their answers respectively [19]. They proposed a neural network architecture that learns through an encoder-decoder model that generates an input-output sequence by reading each word of input to update states of the Recurrent Neural Network approach [19]. In their plan, they match the vocabularies with Glove word-vectors² for the target language lexicon at the first step. Then they implemented a sequence encoder-decoder model to map onto manually built SPARQL templates [19].

Restricted Domain Question Answering evaluation has been tailored from Open-Domain Question Answering. [Diekema, Yilmazel, Liddy 2004] has criticized the test questions developed for a restricted-domain question answering through open-domain question answering parameters [25]. The authors stated that task orientation becomes essential in evaluating a restricted domain question answering. Their data source is

² <https://nlp.stanford.edu/projects/glove/>

plain-text documents in the aerospace domain that includes academic content of aerospace engineering. Chiefly, [Diekema, Yilmazel, Liddy 2004] split the parameters up as performance testing, data source testing, and user interaction testing. The performance testing includes speediness and availability of an answer in responding to a question. While the data source testing was observing the scope, coverage, size, and updatedness of their data source, the user interaction testing specified the testing phase as querying style (Keywords, sentence-based), question formulation assistance such as spell checker, abbreviation recognition and feedback collection [25].

2.3 Linked Data Collection for Heterogeneous Data Source

Linked Data Collection is not part of our research objective, but we will overview approaches for the sake of the semantic question answering. We should conduct a literature review to find out the status of the researches. Linked Data Collection for the assistance software can be examined within two sections.

2.3.1 OPC Unified Architecture Information Model Serialization

Firstly, linked data serialization from OPC UA Servers to convert Information Model into a linked data format. There is a research gap between OPC UA Communication Stack and linked data model, and it appears that researchers' circle have not conducted enough surveys, researches, and statements. Information model provides standardization of information representation to be understandable by the several systems. We will research the serialization of the OPC UA Server Information Model into the linked data.

[Katti, Plociennik 2018] proposed an approach about the integration of OWL linked data language into application-specific methods of OPC Unified Architecture. The emphasized that the absence of a standardized information model for machines, each vendor may implement their data and information model to devices [26]. Their aim to create a semantically augmented OPC UA framework that enhances the knowledge in production for decision-making systems [26]. The authors of the paper defined the essential parts of OPC Unified and OWL linked data structure that should match each other. For instance, they mapped "serviceName" of OWL document onto "OPCUAMethodName" of OPC Unified Architecture. The ontology service creates the ontology statically

through the various process of the factory such as welding machine, color-spraying machine. They have not published test and application details except for a piece of images related to sample created OWL data. [Pfrommer, Grüner & Goldschmidt Et al. 2016] offered a technology-independent common core for information modeling to overcome various information model from different machine-to-machine protocol [27]. The difference of their approach is creating a uniform information model that can adapt the information into SQL Database, memory-mapped values, OWL ontology or AutomationML formats[27]. The difficulty of their approach that a platform should discriminate the containment relations, inheritance relations and type-instance relations of the Information Model in order to assign to particular ontology items [27]. They have cleared the intermediary language before mapping onto a semantic ontology; hence there would be a performance advantage.

2.3.2 Linked Data Collection from Streaming Data

Secondly, we will search over the linked stream data processing to create format suitable to linked data. Linked Stream Data Processing with linked data is the primary research topic in Industry 4.0 and Smart Factories. Previous studies mostly defined Semantic Representation as a challenge that is supposed to map from the time-series data onto linked data. Raw sensor data is useless unless without being adequately annotated.

Previous studies mostly defined the linked data collection from streaming data as a challenge because mapping from the time-series or real-time data onto linked-data creates different nature among them. [Su Et al. 2014] offers a mark-up language for representing device parameters and measurements [28]. The authors organized research for a sensor markup language that used to describe sensor measurements and device to find the gap between semantic representations and data formats [28]. SenML is an intermediary language for sensor measurements, and they convert this language to linked data. [Wang, Zhang & Li 2015] have defined two main rules to implement a semantic annotation, which is transformability to multiple RDF sources such as N3, Turtle and automatic assignment of a namespace to be specified on the sensor and actuator applications [29]. The biggest drawback of this paper that was poorly designed to show how an intermediary language could be converted to another language.

Establishing a way to extract automatically from unstructured time series data into linked data is a challenging problem. [Llanes Et. al., 2016] states that the real-time approaches of linked data suffer from the main limitations which are [30] :

- 1) Triple storage cannot efficiently handle high update rates
- 2) Numeric reading has performance issues with complex SPARQL queries.
- 3) Extracting sensor data triples are different

As being a survey paper, [Llanes Et al., 2016] categorized real-time data for linked data with a selection of ontologies, defining the mapping language, choice of continuous queries, choosing related datasets in Linked Data Cloud Storage and creating data linkages [30]. Each chapter have given a definition and current research in the market, and it can be summarized as below:

Selection of Ontologies: Ontology selection is a crucial step to perform streaming linked data from time-series value. Every platform has its specifics, and it should be handled with proper RDF datasets such as OWL, Turtle, and N3. Lack of scalability from a semantic data source to another, platforms should use standard semantic dataset and annotations. [Llanes Et al. 2016] offers to use Semantic Sensor Networks that can describe capabilities, measurements, and resultant from sensors and actuators [30].

Defining the mapping language: To convert sensor-based data from time series into Resource Description Framework, a converter needs an extra layer to customize mappings from relational or non-relational databases to RDF datasets [30]. [Llanes Et al. 2016] demonstrates two approaches which are: R2RML [Calbimont, Corcho & Aberer 2011][31] and SASML [Zhang Et al. 2015] [30]. Those languages are in common that the platform works with time series streaming value should have a mapping layer in order to send a SPARQL request. While R2RML were implementing a sensor network that contains predefined annotator to match every possible sensor or devices with a framework in particular ontology language, SASML used to annotate sensors and devices in extensible markup language simply.

Selection of continuous queries language: The authors stated that expressions such as SPARQL are designed to execute RDF triples in a static way, however SPARQL query has no effect on streaming linked RDF triples so that new RDF Stream Processors were implemented by [Barbieri Et al. 2009], [Calbimonte Et al. 2011], and [Anicic, Fodor 2011]. They named the new language C-SPARQL and Event Processing SPARQL respectively [30].

[Anicic, Fodor 2011] proposed Event Processing SPARQL (EP-SPARQL) as a new language for complex stream events [32]. The primary goal of their proposal is to provide a

fundamental framework for Event Processing and Stream Processing [32]. The authors created a new quasi SPARQL language that has some similar functionality such as Seq, Equals, OptionalSeq, and equal optional used to combine graph patterns in the same way as Union and Optional in SPARQL [32]. While event processing is adjusting the time window size in SPARQL, stream reasoning organizes the subject-predicate-object triples coherently. EP-SPARQL language can take advantage of query optimization and pre-processing over the static and dynamic part in data space unlike C-SPARQL [32]. C-SPARQL is an older version of EP-SPARQL that consists of RDF streams, Windows, Registration, and Aggregation [33]. C-SPARQL language is less complicated than EP-SPARQL. RDF Streams are locators of data source identified by Uniform Locators. Windows describes many given triples should be in the timeframe. Aggregation and Registration provide similar functions such as bool indicator, average, sum, min and max in the same way in SPARQL.

[Hasemann Et al. 2018] proposed an RDF tuple store named Wiselib that attaches into sensors to collect data employing RESTful architecture that can connect to Linked Data [34]. The Wiselib on the lowest level, it uses a set of protocol that a sensor can understand at the same level. On the highest level, it uses HTTP protocol to understand semantic web documents as a proxy server. As an extra feature, the tuple store can behave as a SPARQL endpoint by basic query parameters such as browse and insertion statements [34]

2.4 Chapter Discussion

Previous researches about OPC UA integration to RESTful API profoundly gives advantages and disadvantages with the implementation details. As of yet, there is no examination to show an architectural design that can be compatible with industrial manufacturing. Except for a solution created with research[7][8], other research algorithm or data set are not reproducible. No research has been conducted overlaps with our research goals in the thesis. We will elaborate given algorithms on previous studies regarding OPC UA-Web Integration, the heterogeneous data source, and the semantic question answering.

First of all, OPC-Web Integration has not been overviewed about architectural design of backend and frontend application in any manufacturing domain. Besides, there are some

meaningful design advice, outlines, and evaluation in the research [9][7][6] [35]. The research in [Cavalieri, Salafia & Scroppi][6] [7] has detailed the Publish/Subscribe implementation of OPC UA. They created a monolithic architecture without balancing, discovery service, and front-end architecture. [Paronen 2015][9] has defined the scope as a practical implementation of supervisory control and data acquisition system, historical data acquisition and monolithic service and presentation layer architecture. Not all the other researches are similar to our research goals; nevertheless, they can help the thesis with their knowledge base of communication stack and possible implications they have found. [Grüner, Pfrommer & Palm 2015], [Grüner, Pfrommer & Palm 2016], and [Shiekofer, Scholz & Weyrich 2018] addressed the problem set at the protocol level, which is regarding network communication.

Serialization of OPC UA Information Model grouped into semantic sensor network-based approach and adapting information model approach. The semantic sensor network is still hard to implement and understand without a knowledge base of the underlying system. It has been a bottom-up approach ranging from sensors to main manufacturing devices. Unlike semantic sensor networks, generating a uniform information model could be a top-down approach, but the system would be generic to every industrial communication system. The thesis handles OPC Unified Architecture so that there will be no needed to apply these approaches.

As for the semantic question answering, each approach can solve the specific domain problems. It is a cumbersome task to analyze every application in open-domain, closed-domain or restricted-domain question answering; instead one can examine the fundamental algorithms. Rule-based and statistical learning based methods come to the forefront depends on the issue that all authors faced. [Molla, Vicedo 2007] offers a framework from other applications to one who wants to develop a restricted-domain question answering. [Molla, Vicedo 2007] listed the essential features of restricted-domain one can consider them while developing a system. [Tirpude, Alvi 2015] and [Nguyen, Kosseim 2004] propose a modular approach such as question processing module, answer ranking module in the making from questions towards answers.

[Dwivedi, Singh 2013] has been specified the algorithms as categorical, which is a quite important guide for existed algorithms. [Tatu Et al. 2016], [Celikyilmaz 2006], [Palaniappan, Sridevi & Subburaj 2018], and [Unget Et al. 2012] have employed statistical methods to reserve morphological structure of a language. Advantages and disadvantages of their approach might not be easily observed because every approach is special to its domain and test set. Significantly, [Ferre 2012] has offered a controlled

natural query that can restrict the grammar structure. The controlled language can eliminate the methods of pattern-based, template-based solutions and it can benefit less resource from statistical language identification.

Additionally, [Diefenbach Et al. 2017] proposed a significant analysis on the techniques Knowledge Base Question Answering, which is quite valuable analysis that can be used in the thesis [17]. This examination consists of most possible use cases concerning with methods. Given headline on the question analysis, phrase mapping, and query construction may serve as a useful functions. The advantage of this paper is that the examination has been covered fundamental situations. However, the disadvantage is that the survey did not focus on restricted domain question answering. In the conclusion part, they deduced that the high recall and low precision can be good or bad depending on the next steps, but they did not correlate which is which.

Consequently, insufficient test data and lack of producible algorithms are perplexing the reliability of algorithms in the domain of smart factory. Another fact is that the algorithm strongly depends on the test and training data; therefore considering requirements of the field can give us a more significant contribution, rather than implementing in the thesis credulously. As for the OPC Unified Architecture with REST integration still has a problem to combine at a different level of network communication. Architectural design and application field of an OPC UA web-platform in previous studies have not given much detail that we can follow.

3 Industrial Communication in Smart Factories

The fourth revolution of Industry as known “Industry 4.0” fostered the exchanging data communication between interconnected devices in an industrial plant. The primary objective of Industry 4.0 was making the manufacturing technologies of factories more intelligent, optimizing the chain of processes and enhancing capabilities of communication one to another. Industry 4.0 enforced end-to-end digital integration of engineering throughout the value chain to facilitate highly customized products, thus reducing internal operating costs [36]. Regardless of being the context of manufacturing, Industry 4.0 has changed communication infrastructure between devices and machinery industrial communication working groups have always aimed in industrial plant requirements Increasing efficiency and productivity that has required. Having a unified digital platform for industrial communication, devices and machinery can communicate with each other regardless of vendor-specific applications. Interactive assistant tools, transformation ability, and efficiency in production are key factors that can affect the future of digitalized factory.

A smart factory is a highly digitized and connected production facility that relies on smart manufacturing [37] — this concept one of the key outcome of Industry 4.0, which intelligently changes manufacturing technologies. The central power of the smart factory is making data collection possible. Additionally, sensors enable the monitoring of specific processes throughout the factory that increases awareness of what is happening on distinct levels [38].

The definition of smart factories has evolved over the past few years. In the present studies, a smart factory has defined an aspect of boosted technology of Industry 4.0. Impact of manufacturing development affected economic growth over the last decade in Germany. Continuously improvement of Industry 4.0 brought the researchers to find cutting-edge technologies such as Question Answering System; Manufacturing Augmented Reality, ChatBot System and so on. The critical aspect of Smart Factories can be list as follows: Towards Smart Factories, Industry 4.0 and Human Machine Interface in Smart Factories. Within the essential aspects, this study informs the readers how it contributed to the Industry 4.0 area and what will be the benefits when used by Smart Factories.

Consequently, it is necessary to integrate the value chain by using cyber-physical systems digitally has been demanded [36]. The Cyber-Physical System embraces complex networking, integration of embedded systems and application systems, enabled by

Human Machine Interaction [39]. Smart operators should be enabled to realize the helper system to humans in increasing interactivity and ergonomics.

A cyber-physical system describes the relationship between humans and a Cyber-Physical System, which is divided into a physical component by separating virtually from each other[40]. Taken as a whole, physical components and their virtual representations should standardize from the bottom to the top.

3.1 Human Machine Interaction and Smart Operators in Smart Factories

Manufacturing is one of the critical areas that should communicate with humans clearly to increase the overall efficiency of a factory. After Industry 4.0 has been initiated, the new industrial development revolutionized by the integration of interconnected devices into manufacturing systems. Although the digitization of manufacturing process has changed the sense of human operating, the human operator still is a core element of industrial automation. The decision makers planned the industrial automation to eliminate the cost originated by human operators. Even though many devices in a plant can handle the processes automatically and robustly, those processes should be optimized and maintained by human operators. The Human Machine Interaction is a term utilizing for the relationship between humans and machinery, which can be managed by a smartphone, terminal device or monitoring device. Moreover, human-machine interaction such a complex system organizes among human operators, process control systems, and people at management in a smart factory.

Actionable feedback through smart devices became significant so that the human operator reaches information anywhere and anytime in a smart factory. Those smart devices can use desktop applications, tablet applications, embedded controller applications or web applications. Continuous monitoring over streaming data, browsing data through vendor-independent devices and natural language inquiring over heterogeneous data are essential skills that a human-machine interaction should provide.

With the improvement in the OPC Legacy Standard, the Industrial World achieved the interoperability between heterogeneous devices at the communications level, regardless of the manufacturer [41]. Question Answering systems increase the capability of transforming query languages. Semantic structured or relational data used to show result employing a particular query language SQL or SPARQL.

The web-based solution has an advantage over developing a smartphone application or tablet application in accordance with scalability. Since a mobile device can reach to a server, the HMI system can connect all device according to the web server's configuration. Hence, the system may enhance user experience with Human Machine Interaction legacy devices in order to interact with the data of industrial processes [41].

A human operator should obtain various kind of stimulus with an assistant application to initiate a situational recognition that conveys to problem-solving step. Such tasks predictive maintenance or diagnosis on sensors or actuators is not possible without knowing the underlying structure. Nevertheless, a smart operator can help to solve this type of issues that human operators faced, beyond that the human operator can inform about situational analysis to the management personnel. Throughout all process may affect the decision-making process or at least informative resource would be obtained in a short amount of time.

In our case, experts may employ the average value of a specific sensor that resides in a machine to predict future maintenance or repair. The system can also provide an error situation with a threshold value when querying into time series data. Because of domain-dependent data, a question answering system complies with the factory specific data. The data may contain many specialized terms that experts use a keyword or plain text to search for an item related to a machine.

3.2 OPC Unified Architecture for Web Applications

3.2.1 OPC UA Service Sets

OPC UA has low-level communication stack and high-level applications on the communication stack. Because non-existence of a set of subroutine definitions for communication stack, services are conforming to their service sets. To implement a web-based application, OPC UA Services must be suitable with regarding web architecture. OPC UA Services have fundamental specifications and requirements for each service. Each service has service sets that show abstract descriptions, and they do not implement specifications [42]. These services could be discovery service set, secure channel service set, session service set, query service set and subscription service sets. Discovery Service set used to find an OPC UA Server endpoint that has connection string by conforming au-

tomatically or manually. Secure Channel service set ensures the confidentiality and integrity by certification exchange among clients and servers [42]. Session service set used to create a session with the secure channel service set to reach an address space of a server. Query Service handles current and historical data concept to return bulk data from an address space. Lastly, the subscription service sets have a set of polling item called "*MonitoredItems*" defined by clients to notify progressive changes in a particular time interval [42]. So far, we have explained the abstraction layers of OPC UA communication stack for higher-level functions. The following parts will introduce the necessary parts that we examine in the thesis.

3.2.2 OPC UA Address Space Model

The primary objective of the address space in OPC UA provides a standard way to the clients in terms of elements of OPC UA. More specifically, the Address Space provides an area to objects that can realize to exchange information. To exchange information, the address space act as permanent storage transforming from binary data to high-level objects. In the very beginning, OPC UA has specified as an object-oriented model, and every element of OPC UA need to correspond for objects. OPC UA had to comply with this standard. Clients can browse, read and write using nodes in the address space.

The smallest item in the address space of OPC UA names Nodes which belongs to Objects [43]. A node comprises Attributes and References which can be reached by Node Class Browse Name [43]. Attributes define Nodes, and a node can connect to other nodes with the interconnected information of References. OPC UA Nodes have several classes such as Object, Variable, Method, View, Object Type, Variable Type, Reference Type and Data Type [44]. When a user is endeavored to obtain values of the node, the address of the node in Address Space of OPC UA should be activated. Mainly, a browse name and node-id show to clients in the address space. To access attributes or other elements, clients must know the name of browsing and a related node ID. Due to a real-time data processor, the address space has a breakthrough feature where process data saved previously. This thesis is a review of a preliminary attempt to explain items of address space which are [45]:

View: All Nodes lives in a View when browsing in Address Space.

Object: It represents real-world objects and software components, and it may use References additionally to define relationships of Nodes.

Variable: The purpose of a Variable is to provide a real-time value when a client is browsing in it.

Method: Method item correspond to callable events by returning a state

The things as mentioned above defined as the general aim of OPC UA Address Space. There must be data containment when we use these items. Mandatory and Optional selection are contained in type definition so that one can decide how to apply a type.

Object Type: It consists of a definition for Objects

Reference Type: This type used for meta-modeling providing an inheritance of objects and defines meanings of a relationship among nodes.

Variable Type: It defines some types such as Historicization of Variables, Minimum Sampling Interval, Access Level, User Access Level, Array Dimensions, Value, and Object Type. Variable Type has a vital role in practical implementation because the definitions of Variable Type enables browsing, reading, writing, and subscription how to make them possible.

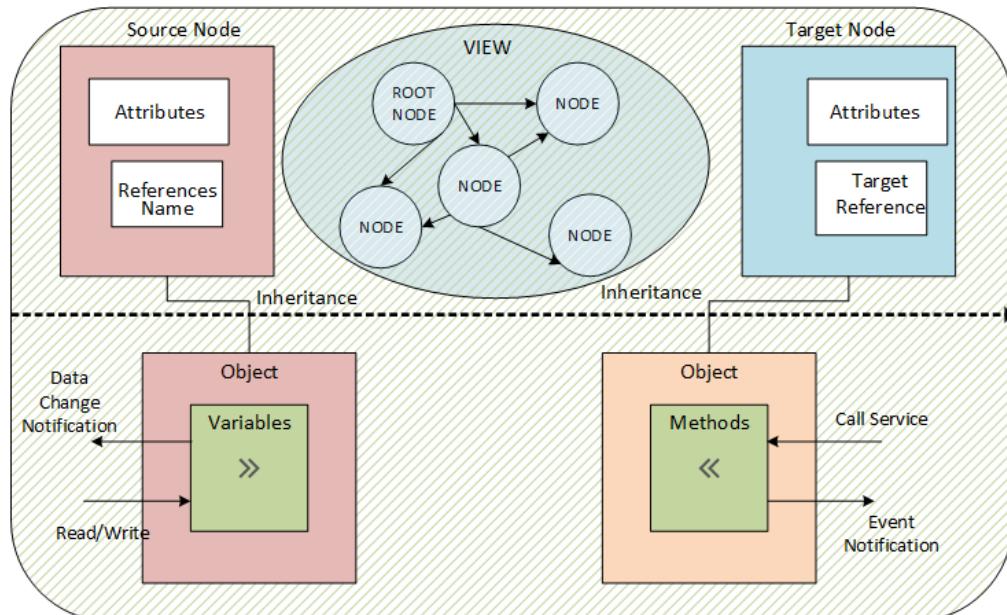


Figure 3.1: OPC UA Address Space [43] [42]

As shown above, an address space consists of nodes. Each node has fundamental attributes and references. Object classes have variables and methods to embody object-

oriented architecture. Nodes connect to each other in an address space through a service called View. Upper space that has separated by a dashed arrow shows an address space and below one demonstrates an information model that inherits from and links into address space. The View Service in OPC UA helps to navigate hierarchical references to search for information about nodes, attributes, and objects of nodes.

3.2.3 OPC UA Information Model

The Information Model coordinates the structure of objects that have relationships with Variables, Methods, and Events and provides a set of predefined types and rules which can be expandable [45]. Beyond this concept, a semantic modeling tool provides a two-way standardized communication version of an Address Space. Strictly speaking, it is a way of object-oriented representation of servers that can be reached by clients. The main difference between Address Space and Information Model is suitable with meta-modeling languages such as UML and SysML. Information Model has a higher abstraction layer to simulate the Object-Oriented Paradigm of OPC UA Protocol.

As indicated previously, OPC UA is a protocol based on Service Oriented Architecture so that every object can communicate related service to exchange corresponding data. Object Types defines types of object dependent on the object, and these types can be customized with multiple definitions. Variables are the main components of objects that represent data values in the objects. Variable Type and regarding Data Type define a structure of variable. The challenge of any OPC UA Software shows all data types that relate to Scalar (Fundamental Data Type), or Application-Defined Data Type. Our method is a definite improvement when a user requires reaching a scalar typed or application-defined typed objects.

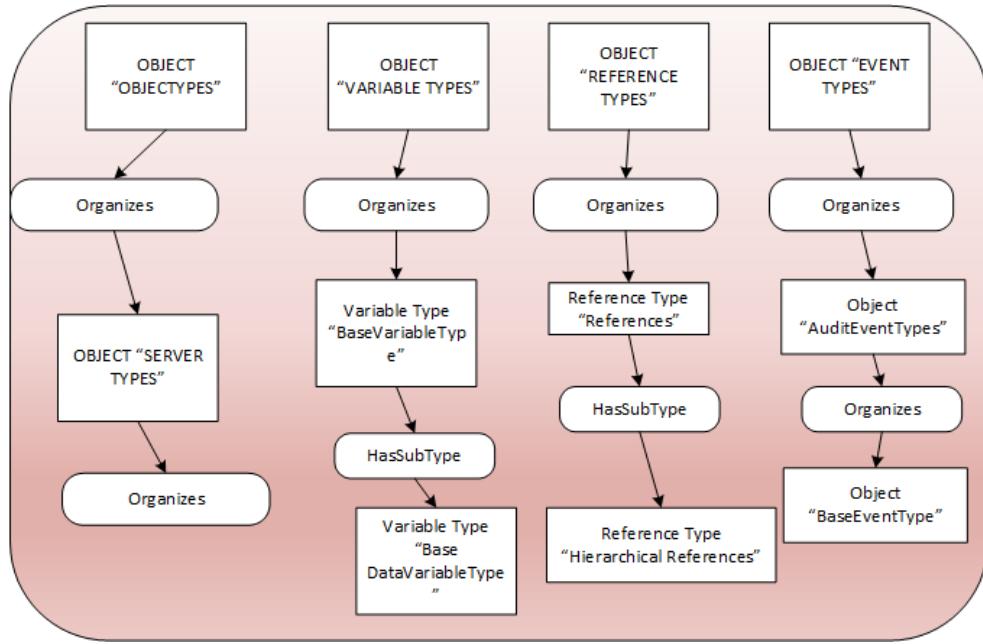


Figure 3.2: OPC UA Information Model [46]

Information Model defines objects, variables, events, and references relationship between them or inner structures. As illustrated in Figure 3-2, a reference organizes the relationship between object, more specifically Node Classes. Node classes are a subset of the abstraction method among nodes in the address space. Due to space limitation, Figure 3-2 indicates limited elements of the Information Model. For instance, reference has not only HasSubType, but also comprises of HasTypeDefinition, Organizes, HasProperty, and HasComponent. HasSubType defines subtypes and supertypes of references. While subtypes are specified explicitly, supertypes are identifying through HasSubType implicitly. For instance, the BaseDataType has multiple references as HasSubType, and with a BrowseName and NodeClass, it is defined explicitly as primitive, structured or XML elements. It is not a mandatory type of definition in references; nevertheless, it is a compulsory schema structure build up a hierarchy in OPC UA. HasTypeDefinition is a definitive term for the type definition of an Object. Every object has a relationship with other objects so that HasTypeDefinition should occur more or less the number of connections that an information model has. Organizes determine types of folders and their internal structures so as to group a set of objects.

Nevertheless, Organizes reference may be used for Objects of the FolderType, which has a usage restriction [43]. OPC UA Servers have useful items called Folders that serve as

separator objects that have similar type definitions. HasProperty used to describe Properties, which properties have relationships with other features of a Referent Type. HasComponent identifies the data variables, the Methods, and Objects contained in the Object [43].

Consequently, every element of the Information Model defines and has a relationship in a hierarchical way or a non-hierarchical way. In a general manner, the information model defines types and references, which are the essential component of abstraction. As a result, the web-based application depends on the information model for browsing between nodes with their reference and for identifying the types of the nodes, more specific objects, by using this service.

3.2.4 OPC UA Discovery Service

The principle of SOA states a service-oriented architecture should have a service consumer, a service provider and a discovery service. The discovery service is vital to construct a microservice structure instead of statically typed endpoints. The practical implementation is able to use discovery service and then clear-cut key points described in service discovery of OPC UA. Overviewing better the discovery process should examine these scenarios as follows. A client and a server can be in the same host or the same network. Moreover, a client can connect different servers, which are in a different network.

In the discovery process of any network, a discovery service allows to locate items of a network by a specified device of a network. For instance, a client can find a server via a proxy server without knowing any details except the address of a proxy server. OPC UA Discovery Services work with the same principle by using endpoints to establish communication between OPC UA Clients and Servers. Discovery Service at OPC UA Standard can be divided into two main topics in terms of application domain where application lodged in. These are “LocalDiscoveryServer” (LDS) maintains discovery requests for all applications if clients and servers are on the same domains and “Global Discovery Server” (GDS) preserves discovery information for all applications if clients and servers are on the remote domains [42]. GDS can be full-fledged OPC UA Server and centrally organize other discovery services. Conversely, LDS can only behave as a service or serve other LDS supporting multicast networking. In this work, a local discovery server is examined in terms of benefits and drawbacks on the existing architecture. A client that requires connecting to a real server through a discovery server should use

a set of service sets which are “Register Server”, “Find Servers”, “Get Endpoints” and “Find Servers On Network” [42]. When a client requires establishing a connection, a session is not supposed to be created. Hence, every server has a Discovery Endpoint to connect clients without creating a Session [42]. However, this could be a security vulnerability because a client and a server do not share certificate among them and lack of a session creates an unsecured connection.

A Discovery Server has two types of endpoints, which are discovery endpoint and registration endpoint. While a discovery endpoint provides a connection to clients, registration endpoint awaits a result from discovery endpoint whether it has a connection with the client or not. After a client obtains a “GetEndPoints” service set from a discovery process, it can open a secure channel by providing a certificate, hashed authentication or anonymous way to perform opening a communication channel. Accordingly, the between finding an endpoint and sending the endpoint request has not authentication schema. Hence a discovery service implementation could cause a security vulnerability inter smart factories.

Architectural Decision	Advantages	Disadvantages
Industrial Communication with Discovery Service	Suitable with Micro-service and SOA Design	Insecure connection before “getEndpoints” between Client and Discovery Server
Industrial Communication without Discovery Service	Suitable with Monolithic Design	Lack of automatically endpoint discover

Table 3.1: Discovery Service Pros and Cons

3.2.5 OPC UA Subscription Service

When a stateless architecture such as RESTful API implemented onto a stateful architecture such as session-based protocol, there would be an issue for identifying alteration of data. As streaming data were incoming from servers, the stateless architecture such as RESTful API has a deficit to refresh data instantly. A simulated data that is continuously

changed has a considerable overhead when sending a read request over again. Moreover, data fluctuation has a vital role in analyzing data by experts. Hence, instead of sending a request, a subscription might have sent to identify a variable, attribute or object changes with a set of features. In order to remedy this repercussion, a subscription is sent with particular monitored items into a session and monitored items serve as a polling mechanism. As illustrated in Figure 3-4, a single monitored item and multiple monitored items attach to a subscription. This service reduces time and space complexity of reading request by showing all changes in a single subscription. Three types of changes can be observed in OPC UA Protocol to simulate data, which are data changes of Variables, Objects of Events or Attributes. The sampling interval is a critical component of monitorable nodes to detect changes in a particular polling time. After assigning a sampling interval, OPC UA Server can notify OPC UA Client when an Attribute, an Object or a Variable has changed. The implementation of web-based application dispatch a binary indicator belongs to monitorable nodes; thus the web service sends a general subscription request without monitoring nodes' topics and ID. A filter decides whether the next notification of a subscription should send or not. A filter can eliminate a different type of item to be monitored so that unnecessary notification cannot overflow in the system. A subscription service put all notifications into a queue that can transfer without blocking respective notification.

If a new notification has been entered to the queue, a prior notification should be deleted to free the queue size. Monitored items should comply with the minimum sampling interval. As a result, the minimum sampling interval defines the degree of the sampling interval, and this minimum value of the sampling differ from a node to another node. However, the underlying structure of the update cycle is not synchronized, so the system should explicitly synchronize all sampling values in order to fetch correct notifications with a decent value. Accordingly, the amount of the smallest minimum sampling interval can create a maximum load of traffics for OPC UA Server and lead to produce buffer overflow,

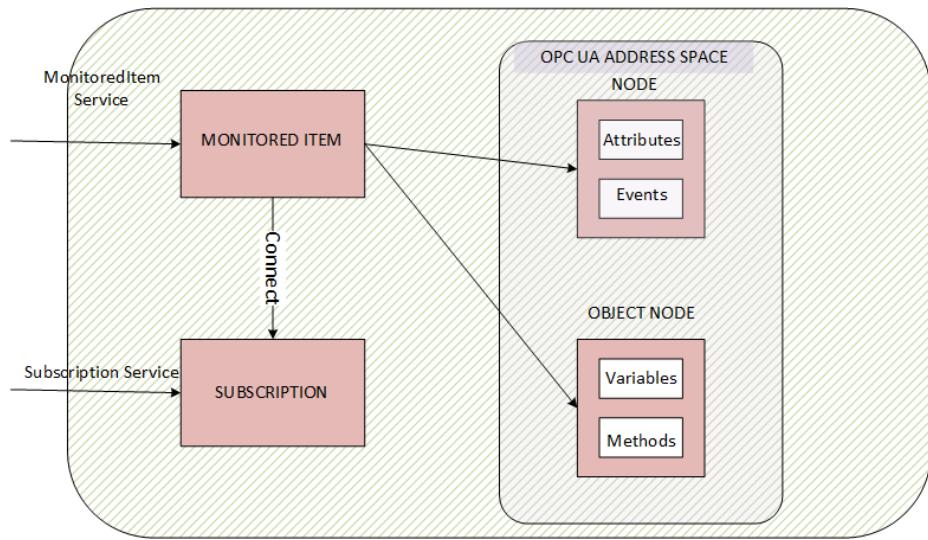


Figure 3.3: Subscription and Monitoring Item Services [42]

3.3 Chapter Discussion

Components of OPC Unified architecture have referred to data exchange structure, their definition in communication stack and abstraction methods. OPC UA Servers and Client are the fundamental elements of this research. Due to insufficient information modelling of communication stack of OPC Unified Architecture, each OPC UA Servers realizes its information modelling and address space. Moreover, data that has stored in OPC UA Servers have various definition and it can change the semantical understanding.

External connections of primary functionality of OPC Unified Architecture organized with OPC UA Service Sets.

RQ1, RQ2, RQ4 should be answered.

4 Theory of the Semantic Question Answering over Linked Data

The question answering is a balancing process between natural language understanding, information retrieval, and information extraction theories. The description of question answering, though appearing so easy, in essence, is a broad field of research with categorized several headlines. Principally, a question answering enables simple query rewrites without explicitly the use of a query language. On the one hand, a question answering performs a task on natural queries to detect syntactically and semantically; on the other hand; it is an activity to obtain proper information model that has been searched. Therefore, the principal aim of a semantic question answering is to identify an answer from a bunch of collection that can be ontologies or documents. Since that Information Retrieval and Information Extraction have perplexed definitions that one should examine with similar and dissimilar points, to better understand the question answering process. Information Retrieval is a research term used to locate a document that required by a user, but a user defines a relevant answer after obtaining a document. Information Extraction is a term for extracting a set of information from a user input so as to learn relationships between searched keywords and documents³.

According to the functionality of Question Answering:

Information Retrieval-Based Question Answering has a goal to answer a user's question by segmentation of typed contents of text on collected documents [47]. Main features of the question answering are strongly bounded to answer extraction module. It means that a document similarity algorithm such as TF-IDF should be realized after obtaining answer sets.

Knowledge-Based Question Answering has a rudimentary idea that subject-predicate-object triples of ontology is able to map onto simple relations of natural language queries.

³ <https://www.ontotext.com/knowledgehub/fundamentals/information-extraction/>

According to the data source:

Ontology-based Question Answering is a type of question answering that takes ontology or any other linked data source as a data source. This question answering does not include plain-text documents for data source.

Context-Based Question Answering handles document-based dataset to respond questions asked by users. This question answering does not interest in ontology data.

According to the type of domains:

Open Domain Question Answering: A user can ask any topic that he wants to reach a result from a general domain. This question answering may have a context-based or ontology-based data source.

Closed-Domain Question Answering: A user can only ask questions against domain-dependent document-based architectures. For instance, one can ask general questions against a specific text document which have collaborated by specific domains.

Restricted-Domain Question Answering: It is more likely one can ask a question against semantic documents in order to obtain results. Main characteristics of restricted-domain are that data sources can be different from closed-domain and open-domain. In this domain, the answer and result sets are circumscribed and complex, so information retrieval capability is strictly relevant to natural language processing capabilities.

Question types handling is an essential step for any question answering. On the one side, closed-domain and restricted domain question answering systems used for eliminating the unrelated kind of questions, on the other side open-domain question answering system takes kinds of questions to formulate with rule-based architecture.

According to types of questions will be shown as below:

Factoid Questions are about providing concise facts. For instance, "What is the population of Berlin?" is a factoid question that should be narrowed down from a general topic into a specific one. Otherwise, an open domain question answering system would be ineffective against a factoid question. Factoid questions may include list questions, wh-typed questions or keyword-based search. **Indicative Questions** is that an expert or human operator can make a sentence through request words in the sentences, e.g. "I would like to know what does linkedfactory contain?". A **keyword-based query** is one of the basic search items that are used in search engines. In the early phase of the Internet,

researches have been focused on how to extract documents from keywords. The simplicity of grammatical and semantical structures, a keyword-based search has always been a prominent topic for question answering. This work can use a keyword to extract information from the Turtle RDF data format with a specialized keyword. Verbs or more specifically predicates and objects counterpart nouns in RDF are based on Fraunhofer IWU's data source. So one needs to have limited information about the internal system.

Indirect Requests states a question like "I would like to list all of the members in linkedfactory?" or "Give me the value of sensor1 in machine1". The main feature is listing of the desired results.

Boolean Questions should have answers might be yes or no. We have used this type of question to understand system status. For instance, "Is the system health good for sensor1 in machine1?"

Non-factoid Questions are the type of questions that have not concise facts. For instance, "did you perform well the task?" is non-factoid question because there is reasoning and induction to find out correct answer. These questions can be grouped as reasoning questions. A user may ask a question defining by reasoning keywords, e.g. "Can you tell me the system health in trouble?" or "Can our system stay alive?". "Why" and "How" questions also show reasoning to induce a result from a series of events. The main reason that this research did not implement is that the types of questions have complex structure for statistical model, require a vast amount of data, and answer ranking module.

Lastly, the proposed semantic question answering can be considered that is a knowledge-based question answering in terms of functionality, which has an ontology-based data source that can answer factoid questions in a restricted smart factory domain.

4.1 Semantic Web Technologies

Knowledge Representation is a subdivision of artificial intelligence used to understanding and interpreting data in representing methods of computer language, so that unrelated domain can solve the problems of external representation. For instance, natural language understanding can match the vocabularies with triples of ontology in semantic web technologies. This subsection examines the semantic web technologies in the context with knowledge representation.

Resource Description Framework is one of the most ubiquitous data model for interchanging web-based information. As it is understood from its name, it is a framework for supporting resource description and metadata in the linked data. Each RDF member represents as triples and each triple might have connected to other triples. First RDF version provides a set of features that can be used interoperably with the extensible markup language (XML). The authority of W3C controls RDF specifications in terms of update and maintenance of new requirements⁴. RDF consists of several types of models that currently used in industry. The fundamental part of an RDF data is a prefix so-called Uniform Resource Identifier so that query languages navigates internal structure with URI. Resource Identifier is rarely not feasible to every generated document from an extensible markup language. The primary purpose of the Resource Description Framework is to be a machine interpretable medium for linked data in the world wide web. Algorithmic representation of RDF is a graph data structure, which has a set of vertices and edges.

Serialization stands for converting an RDF Format to another to use a variety of syntax notations so that the particular encoding can produce a variety of triples. After serialized an RDF resource, one can obtain the following formats. RDF states that URI Abbreviation is suitable and concise with namespace rules. For example, a blank node namespace defined by an underscore, but regular namespace should not use that underscore. XML Schema primitive types must be compatible of RDF types. In the case of incompatibility, there might be increasing blank nodes or undefined type values, so it would degrade the feasibility of a question answering.

Besides **Turtle** being a reliable alternative for RDF/XML, the syntax of Turtle Semantic is similar to SPARQL queries. Turtle Notation is a compact and clear structure. Predicates and subjects can be marked as a block. Each termination of <subject, predicate, object> triples end up with a dot character and sub-triples are connecting to other sub-triples with semicolons. RDF serializers can translate this language to other formats without handling complicated Unicode characters. The practical implementation has used this format to adapt data integrity with less overhead while parsing.

Notation 3: N3 triples are similar to Turtle RDF unlike it is supporting underscored namespaces. N3 triples syntactically is a subset of Turtle RDF because it was designed to be a simple format than Turtle RDF. As much as there are similar syntactic definitions, a variety of differences unlike Turtle RDF has been observed. Triples follow the pattern

⁴ https://www.w3.org/standards/techs/rdf#w3c_all

<subject-predicate-object> and a terminal notation. Notation 3 has enlarged grammar structure with extra features more than Turtle RDF and NTriples.

N-Triples Parsers and serializers can easily parse this format because of simplicity. There are no complicated grammar rules with N-Triples, but it is not a suitable format as human-readable. N-Triples has a trade-off to increase machine readability over human-readability. The most straightforward triple statement is a sequence of subject-predicate-object containing white spaces and dot-separated values. It has a tedious format has not abbreviation feature that makes hard to read by humans.

JSON-LD provides a lightweight linked data format, so objects should be converted to a human-readable format. This format is a compact format that has compliance with the JSON data. JSON-LD format has a compact dependency on JSON format, and it can be used without prior information about RDF. Typically, JSON-LD contains the same structure as compared to RDF like primitive types for nodes and IRIs definition for edges. Standard parsing methods for JSON can be used for JSON-LD interchangeably.

//Explain types, aliasing, nesting, language

Web Ontology Language (OWL) connotes a bright and compact way among relationships of data. Prefixes with IRI is one of the fundamental structure of OWL linked data. OWL can define a class to provide abstraction within the same linked data document. OWL definition is a standard with a Prefix IRI such as “<http://www.w3.org/2002/07/owl#>”, and it is a mandatory field to define if an OWL source used in a linked data. This ontology language leverages RDF Schema to recognize complex knowledge requires complex properties [48].

//What is the benefits of OWL – Why did they invent it?

SPARQL Query Language is a specialized query language by using a particular protocol endpoint for performing data manipulation from RDF datasets. RDF has a collection of graphs, and these graphs are directed and labeled. As a result, triples of graphs can be obtained with a query language from databases or files. The structure of SPARQL resembles Structured Query Language (SQL) very much, but the SPARQL was designed using for semantically structured triples, not for relational datasets. Additionally, SPARQL is a definition of a protocol working with HTTP Request by defining “User-Agent”, “Content-Type” and “Schema”.

PREFIX, SELECT and WHERE are three basic operators of SPARQL Protocol. PREFIX makes the serialization steps easier referencing IRIs. Prefixes are used for abbreviating of IRIs in a query. “SELECT” and “WHERE” statements used to find the location of objects. IRIs has a wider range of characters to be used in order to accommodate a wider range of languages than URIs [49].

Mainly, SPARQL Requests characterized by Remote Queries or Native Queries. Remote Queries define as sending a query against a remote SPARQL endpoint. Remote Queries needs an endpoint definition provided by Linked Data Source. As for Native Queries, they work mostly in a local database such as graph databases or files and require a query processor to carry on a query against local sources.

The SERVICE keyword reduces the complexity of queries and hands the complex query duty over the SPARQL Service. Real Time Data Annotation Service “KVIN” uses this keyword to prevent making a complex annotation by users.

Sometimes one needs to fetch multiple values in one single query with integrity known as Federated Query. UNION statement can help at this situation to provide federate property into queries. What can be seen is the working principle of UNION is similar to Outer Join in SQL. It takes all Cartesian Product Multiplication, so one can state that the result of an answer impact issue of redundancy. To reduce redundancy, the UNION can be used with an OPTIONAL statement or query can be optimized with only an OPTIONAL statement. OPTIONAL used for allocating a particular portion of SPARQL into results of triples. OPTIONAL reduces redundancy of data and gives every match in any triples. It is a common usage OPTIONAL query with FILTER that allows measuring up a couple of criteria.

One of the most significant problems is searching a triple, which is a blank node, among triples without clear URIs information. A generated Turtle RDF may define blank nodes that have no clear identification. An RDF serializer can do a preliminary process assigning a traversed property to unclear identification whilst using with a SPARQL query. A traversed property is a linkage between two properties to connect triples each other.

As shown in Appendix A.6, meaningful predicate names are crucial steps to employ with SPARQL queries. Converting from natural language question into triples, verbs often are mapping into predicates. Predicates are also edged labels that connect two nodes in the graph data structure. A missing predicate of a node stands for a blank node. A blank node is one of the evaluation methods whilst creating semantic data. Although

a blank node is not a single evaluation method theoretically, nevertheless it is an essential measurement to evaluate for applicability of question answering with semantic data. Unknown namespace

In this study, SPARQL queries used with Turtle Data Source. The following SPARQL query has been used to fetch triples from generated data.

```
""" SELECT DISTINCT ?property
  WHERE {
    ?s ?property ?o .
    OPTIONAL { ?s ?p rdfs:label. }
  }
  """
```

Listing 4.1: Sample SPARQL against a generated local source

4.2 Data Preparation for Heterogeneous Data Source

The heterogeneous data source has two kinds of linked-data. One of them is key-value mapped streaming data that contains time-series values incoming from eniLINK and regarding hierarchical data. The latter is objects-references mapped static data incoming from a particular OPC UA Server.

This work utilizes an API that has implemented as a service known as KVIN to send a SPARQL request into a specified endpoint. SPARQL endpoint should have a validator so that a SPARQL endpoint process a request that is wrapped up by SPARQL validator. This service is an internal development with an InfluxDB time-series database and LevelDB key-value mapping database. Continuous SPARQL Service was examined for streaming data was examined in Section 2.3.2 is analysing architectural differences in the research circle.

Using a continuous SPARQL language instead, KVIN can map semantic data with properties. A predefined namespace has added in KVIN Service to convert SPARQL triples compactly. Relationship with other components of KVIN can be observed as seen in Figure 0.3. KVIN Architecture maps the continuous values onto graphs in linked data. Then, a key-value graph database has been used to connect subjects and objects through properties. Unlike relational databases such as MSSQL and hierarchical database such as LDAP, there is no primary key or primary node relationship between objects. Nodes

may have properties and relationships to traverse from a randomly select start node to the end node. The execution time of a query increases proportionally according to size the path of traversing not all size of a graph in the store. This is one of the biggest advantages of a key-value database over relational or hierarchical databases.

[50] [51] [45] [52] referenced works have contributed to implementation of serialization process from an OPC UA server. However, the implementation and the algorithm as shown in Appendix B.6 have some conceptual differences to understand better the context. Serialization of OPC UA Information Model has several stages. At the initial stage, nodes of the address space should be saved into namespace array with namespace index. A namespace index has URI and Node ID to be registered as nodes. Import Nodes function traverse from root node until reaching terminal nodes. Build Node Tree takes nodes with namespace index and Node ID by clearing duplicated elements.

“appendXML” function converts all nodes and regarding namespaces into an extensible markup language and then all tree structure is saved into an XML file. Hence, conversion to RDF/XML is an essential step to get a Turtle linked data. XSL Transformation language can trim the schematic structure of XML and stave off the blank nodes as possible as it can. Blank nodes have not URI information , so other nodes have not direct access to any blank nodes [52]. Consequently, XSL Transformation employs one-to-one mapping technique that turns structure of elements in xml schema such as <xsd:attributes> into <rdf:Property>, <rdf:Value>, <rdf:subject>, and <rdf:object> appropriately [52].

4.3 Natural Language Processing in Question Answering

The Natural Language Processing has two-folded, which are Natural Language Understanding and Natural Language Generation. Natural Language Generation is a concept to create from a language description to another description that may constitute a set of formal rules, rules of syntax and semantics. For instance, a machine translation system provides a language exchange interface to perform a set of linguistic rules by words and sentences in transforming into another language. The scope of this work does not address Natural Language Generation; consequently, this work examines methods of Natural Language Understanding.

Natural Language Processing has a crucial role in Human-Machine Interaction Systems. It enables computers to understand a natural query or voice input without formulating any computer language in the form of binary representation and for computers to allow

communication with humans with their language. In this thesis, varieties of methods have been used to extract morphological elements of sentences and identify the main items of natural queries.

Natural Language Processing starts examining a corpus. A corpus stands for the body of texts or collections of documents. Multiple sources of collections named corpora [53]. Generally, restricted-domain question answering works with collections of texts can be from books, manuscripts or offline-scripted sources such as electronic publications. A question answering system that works under a restricted-domain should be good at making clear the complexities of natural language word-sense disambiguation by using methods of natural language processing.

Statistical Natural Language Processing explores a statistical and model-based approach with corpus-driven data sets. This study will use main methods of NLP such as Part-Of-Speech Tagging, Syntactic Parsing and supply supervised learning methods the SVM and Logistic Regression in terms of question classification.

The following listed methods have been implemented in the semantic question answering.

Normalization used to eliminate lowest frequently used words with regards to applications. Unnecessary words and punctuations are supposed to be eliminated seeing the way clear to making a less-overloaded application. Every corpus or any data sources should be refined before a process has implemented in natural language understanding. For example, stop word removal is a normalization process. Normalization includes stop-word removal, tokenization, stemming, and lemmatization process.

Stop-word removal is one of the most common tasks in NLP across different implementations to simplify the input structures given a set of rules for stop-words. Stop-words have a different and unique aspect of every language, so libraries of NLP should provide a new stop-word list in every different language. For example, NLTK has a large of the list for stop-words while using the English Language so that this could bring the NLP a drawback that makes usability lower stop-word sources from one language to another.

Tokenization breaks a text into smallest linguistically significant elements such as word or phrase. Tokenization is a practical part of language modelling, and language modelling plays a big role to decide how efficient normalization would be possible.

Stemming and Lemmatization are normalization steps that cleanse unnecessary prefix, suffix or other morphological appendixes. <Subject-predicate-object> should map onto noun-verb pairs in order to create a SPARQL query. If a predicate has a prefix, e.g. "If:contain", given verb are cleared surpluses by implementing a lemmatization. Usually, stemming can clean suffix, prefix or influx from nouns to reach the pure version of a word. However, a restricted-domain question answering has individual words with suffixes that can have a different meaning for the system, or these special words can belong to a different hierarchy of a tree. Hence, a lemmatization and WordNet synonym analyzation should focus on verbs. The stemming algorithm is easier to implement and faster to process than lemmatization. Stemming can employ a rule-based heuristic approach such as regex-methods. Whereas lemmatization needs canonical form of words in dictionaries. Distinction between lemmatization and stemming is that the lemmatization should be aware of context in given elements, so it can distinguish the meaning of elements according to part of speech tagger (verb, noun, adjective etc.) in different context. For example, "incorporated" is a past tense verb and it should stay in the similar context as verb. Whether staying in the same morphological context or not, stemming only clear the surpluses of the past tense verb. The result of lemmatization for Lancaster stemmer, porter stemmer, snowball stemmer, and a lemmatizer is "*incorp*", "*incor*", "*incorpor*" and "*incorporate*".

Language Modelling defines the overall performance of natural language processing methods. Different types of applications that benefit from natural language processing utilize n-gram language models such as spelling correction, machine translation or speech recognition. N-gram defines the size sequence of a given input. For instance, one can tokenize "Could you give me the average value of sensor1 in machine1?" as "Could you", "give me", "the average", "value of", "sensor1 in" "machine1, ?". Therefore, we can call the gram model as mentioned earlier as bi-gram modeling. Because every output of tokenization is parsed as a two-word sequence. N-grams does not only parse inputs with sequences but also it calculates the probability of each sequence. N-gram defines the scope of analyzation to given a specific language. For instance, if an application requires deepest language property, a natural language system should parse as small as possible to model sequences. So, the questions is "how will a natural language processing method will decide the smallest size would keep diving it?"

$N = 1$ (Unigram) has 20000 parameters in order to so. Respectively, $N = 2$ (bigram) has $20000 \times 2 = 400$ million, $N=3$ (trigram) has $20000 \times 3 = 8$ billion, and $N = 4$ (four-gram) has

1.6×10^7 [54]. Apparently, the more n-gram model we have, the more complex system a question answering system that need to solve.

Furthermore, an input can be dispersed to more substantial sequences, but the context of modeling would be messy. So one can say that language modeling is very relevant to application-specific. By using the chain rule formula, the n-gram model predicts the conditional probability of the next word [47]. As depicted in Figure 5-1, language modeling can be estimated with Maximum Likelihood Estimation.

$$P(w_n | w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})}$$

Figure 4.1: Maximum Likelihood Estimation [47]

For instance, a sentence like “I would like to know where the error is.” represents a probabilistic method as Maximum Likelihood Estimation with $P("I") \times P("would | I") \times P("like | I would") \times P("to | I would like") \times P("where | I would like to know") \times P("the | I would like to know where") \times P(error | I would like to know where the") \times P(is | I would like to know where the error")$. The main problem of this approach is to calculate the long-chain probability of total length. As the size of the sentence grows, a system needs more processing time for the calculation of probability. Chained probabilistic calculation in statistical natural language processing suffers from counting many possibility of complex sentence.

On the other hand, the Markov Model can say the last few words affect the order of the next few words. Markov Assumption concerns $n - 1$ number of words in an n-gram model, but this assumption does not concern from that further. N-gram language models help the creation of corpora. While creating a language model, testing and training data sets evaluate the correctness of language model. In the practical implementation, libraries can assess the corpora through the n-gram model, so the libraries can produce better results in the statistical natural language processing.

Therefore, the next question is about natural language processing how to evaluate n-gram language modeling. Extrinsic and intrinsic evaluations mainly used in the phase of evaluation for language modeling [47]. The extrinsic evaluation stands for end-to-end testing by performing all the system functions over again. For example, if we want to assess the performance of a language model in a software library, the system can be performed multiple times to see the results. However, it takes enormous time when a corpus

is big enough especially, four-gram or further. The intrinsic evaluation separates the data set into a training and test set. The intrinsic evaluation is close to current applications in natural language processing because a test set can evaluate a training set without extra necessity of data.

Perplexity theory implies test set may tell how the given model predicts the results well. It is a measure how well a modelled language predict next word of an item

The more results truly predict the lower perplexity a natural processing system can get. The lower perplexity denotes a better model. As shown in Figure 4.2, perplexity shows an inverse probability of a model. At some conditions, dividend goes to zero value in case that a test set could not be matched in a training set. In this case, perplexity cannot be evaluated. Additionally, there is a possibility that a machine learning approach occasionally suffers from overfitting issue. If a training phase have occurred more than average, a system would not give the right results given a test set and behave like generalizing every test set. Consequently, statistical methods established the regarding information that need to be extracted. They decides how to evaluate probability of result with perplexity formula. Minimization of perplexity helps statistical methods such as tagging, parsing, named-entity recognition achieving precise results with their test set, along with reducing the error of training set.

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_{i-1})}}$$

Figure 4.2: Perplexity formula of a language modeling [47]

Part of Speech Tagger: A sentence consists of a couple of structure including words like noun, verb, pronoun, preposition, adverb, conjunction, participle and article that are main categories of part of speech processing [47]. Part of Speech Tagger mostly employs a markov chain algorithm that is a part of statistical natural language understanding. As previously states, markov model stands for a state can depend on a previous step, but there is no dependency on states of historical steps more than one. For instance, a noun or a verb tells us about its neighbors, e.g., nouns are preceded by determiners, adjectives, verbs [47]. Another example could be like a chess player makes a movement according to the last movement of a rival rather than guessing from the first movement of the rival.

In this step, pre-saved corpora which has a million words has to be annotated by POS Taggers. One of the common list that has an identifier for POS named as Penn Treebank. A treebank used for annotating syntactic and semantic structure of a sentence with million words of part-of-speech tagged text. Selection of a corpus equally important to achieve a result with a parsing process.

A concern of the Penn Treebank is to provide multiple syntactic bracketing if necessary [55]. Multiple brackets are important for example Brown Corpus tags “one” and “the one” as Cardinal Numbers but it “the one” case could be an important determiner in any sentence. Every tagger named as labels, which are clause level, phrase level, and word level taggers. However, it is important to annotate as a common noun (NN) for detecting the head of a noun phrase in a sentence. So “the linkedfactory” and “linkedfactory” are assigned as a common noun or an adjective phrase but those could be identified differently with tagger according to Markov Model of the item in a sentence.

Parsing: POS tagging does not interest in relationship between tagged elements. A tagged element could coherent with other tagged element, which is solving with parsing methods. Parsing methods are grouping the tagged elements syntactically and POS tagger could be though as tokenization method of parsing process. As a natural query is given, a question answering system should understand the grammar behind it. POS tagger is not enough to identify a grammatical structure for complex natural queries. Relationships among noun phrases, adjective phrases, adverb phrases, and verb phrases should be examined in order to map subject-predicate-object triples correctly in linked data. The approach of parsing separated into two main sections, which are the rule-based approach and the probabilistic approach [56]. The rule-based approach is a top-down approach to solve problems via predefined rules such as the way of Regex-parsing. Therefore, a question answering system should define rules precisely to get the correct answer. Open-domain question answering systems use this approach because of the complexity of the bottom-up approach and broadened question types. Nevertheless, a rule-based approach could give undesirable results in restricted domain question answering or semantic question answering and could be time-wasting parse approach. The probabilistic

Syntactic parsing commences parsing sentences with chunking that is a shallow parsing without analyzing the deepest node of the parsing tree. Items can be assigned as a noun phrase and a verb phrase. In our case, this method could be practicable, for instance,

“linkedfactory” keyword might be combined as an adjective “linked” and a noun “factory”. If the parser went into the most rooted leaf, it would have been relatively faster operation.

Various types of probabilistic parser have been prevailed since the natural language processing research started. It depends on the grammar of the English language and how much profoundly information source that required by a question answering system. Formal Grammars of English defines a constituency parse approach, which can identify noun, verb or adjectives in a big chunk like shallow parsing. This approach eliminates of item relationship among nouns, verbs, and adjectives by providing an abstraction method. If a question answering system needs a relationship between subjects and objects, a constituency approach is not suitable to utilize because of shallow parsing. In the case of syntactic parsing, the task of recognizing sentence and its grammatical structure [47]. Syntactic parsing suffers from “*word-sense disambiguity*” problem. This problem denotes that a word can represent different meaning in the sense of location in a sentence. For instance, “*What does linkedfactory contains*” could be differentiated “Could you give me the members a factory which has linked?”. Both sentences are semantically similar but hard to recognize by lemmatization and sentence similarity methods.

Dependency Parser and Constituency (Phrase) Parsers: Phrase Structure Grammar defines the constituents and their relationships with other constituent in a sentence.

A constituency parser likely known as a phrase parser that has an objective is to check the grammatical structure of sentences by parsing the chunks of morphological structure. The constituency parser may not handle the relationship among language items. Constituency parser utilizes the Penn Treebank ⁵ which is used for evaluating as test set by calculating perplexity of probabilistic phrase. Dependency parser analyses the grammatical structure of natural input to define the relationship between the root word and the rest of them. Dependency parsers use universal dependency.

⁵ https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

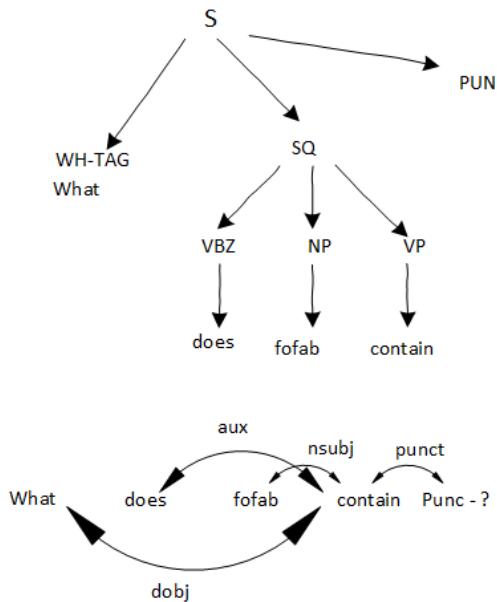


Figure 4.3: A case study of the constituency and dependency parser

Constituency parser may cause disambiguity problem.

//Constituency Parser has a lot of disambiguity

//dependency parser has different stuff

//Talk about shallow and deep parsing

//Talk about syntactic parsing

Spell Checking and Abbreviation Correction: Spell checker is an evaluation criterion for restricted question answering system. It is not necessary to provide advanced spell checker controlling all aspect of morphological, semantical and syntactical rather preferring at least a simple checker. Industrial based spell checker is hard to implement due to some restrictions such as

As for abbreviation correction, it is difficult to find an acronym because of punctuation at the end of the acronym. Domain dependency could be another issue such as computer science, medical, or currency domain. Types of domain mainly used in open-domain question answering system due to a variety of questions. To infer an acronym, a system

expects to utilize a well-formed dictionary overlapping the domain of semantic question answering. A smart factory entirely has different vocabularies and acronyms than a medical domain. In this case, the best way to classify accurately an acronym using a simple look-up dictionary or hash table. A Bayes Theorem and Levenshtein Distance Algorithm would be useful to find both on spell correction and abbreviation checker. However, the spell correction gives better results than the abbreviation checker does under the Bayes Algorithm (given a result set)

Named Entity Recognition: It is a subtask of information extraction to locate and distinctively named entities with pre-classified labels such as names of people, organizations, locations, quantities, etc. Named-entity recognition is a method that identifies the item of a sentence as a domain-specific. It defines all structures mainly as a person, a location, an organization, and an entity. As shown in Figure 5-3, “sensor1” and “machine1” named as an entity and found a relation between each other.

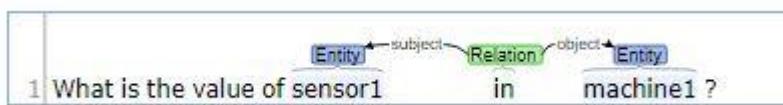


Figure 4.4: Named-Entity Recognition by Stanford CoreNLP

This evidence shows us named-entity recognition is an application-specific task. An NER Method that is created for a different domain may not be reused for another domain. To create a named-entity recognition for a smart factory, a model can be trained to satisfy the requirements of a smart factory. In this context, a model can be created by statistical methods or a rule-based model. In context with the rule-based model, the character regex method can identify the structure of a natural query. For instance, a named-entity recognizer can employ a model that contains a combination of “Heat-Meter” or “HeatingWater”, which it can assert given items with the character started by “Heat” in a smart factory.

Word Vectors (Word2Vec and Glove Data): Words can be represented as vector spaces. One can create the meaning of words table by converting a word into a vector. For instance, if we discuss two main phrases such as “*internet of things*” – “the network of physical objects with electronics, software, sensors, and connectivity”, “*Mesh Network*” – “The topology of a network whose components are all connected directly to every other component”, these two phrases similar in terms of frequency matrix of their meanings. “connected” and “network” are equal semantically. Therefore, one can create a word

vector from corpora in order to identify word similarity. Due to the data size of corpora, a word vector can reduce the feature space of corpora. However, word vectors are not efficient against non-existent word in a vocabulary. Another issue comes out with Sentiment Analysis with Word Vectors. Due to closeness question phrases and negative phrases in some context, those phrases make more complicate the positive, negative or neutral sentence analysis.

Sentence Similarity: Sentence similarity used for comparing two string inputs in order to achieve indicative questions like “Is the system health good?”. Mainly, this method leverages averaging word vectors such as word2vec or glove implementing Euclidian and Manhattan Distances or Cosine Similarity algorithm. In order to calculate distances of word, n-gram model or more specifically bag-of-words concept can be implemented. It is a subset concept of n-gram modeling. In practice, every single element is assigned into an array, for instance when comparing the following sentences:

Jaccard Similarity: This algorithm uses a procedure to calculate the similarity between sets of data defining as the size of intersection divided by the size of a union of two sets [57].

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Figure 4.5: Jaccard Similarity Formula [57]

Jaro Winkler: This algorithm calculates transposition of matrix t, and the number of common characters by putting into a formula as below:

$$sim_{jaro}(s_1, s_2) = \frac{1}{3} \left(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c-t}{c} \right)$$

Figure 4.6: Jaro Formula [58]

The Winkler algorithm increases the Jaro similarity employing initial characters and gives a similarity measurement [58]. For example, Jaro Winkler takes head characters of a string such as “health” and “heal” to perform the Winkler formula.

Levenshtein: Levenshtein algorithm has a variety of application areas such as spell checking, acronym finder or sentence similarity. This algorithm calculates cosine distance of given two strings and divided by the maximum value of absolute value of given two strings.

$$sim_{ld}(s_1, s_2) = 1.0 - \frac{dist_{ld}(s_1, s_2)}{\max(|s_1|, |s_2|)}$$

Figure 4.7: Levenshtein Formula [58]

WordNet Analysis: WordNet is one of the largest databases for English lexicon that can be used for word and sentence similarity analysis. Depends on the domain of question answering, the WordNet Analysis could be used for sentence similarity or verb-noun analysis. In essence, it is a combination of two major algorithms known as Wu-Palmer Similarity and Leacock-Chodorow Similarity.

Wu-Palmer Similarity (wup_similarity): This measure calculates relatedness by considering the depths of the two synsets in the WordNet taxonomies, along with the depth of Least Common Subsumer [59]. With the following formula as shown in Figure 5-7,

$$\delta_{\text{Wu_Palmer}}(c_p, c_q) = \frac{2d}{L_p + L_q + 2d}.$$

Figure 4.8: Wu Palmer Formula [60]

After defining Least Common Subsumer, which is a tree-based semantic relatedness measure extracting from “is-a” relationship of a tree. For example, “contain” and “incorporate” synsets are identical according to Wu Palmer algorithm. First of all, WordNet finds the first Tree with categories like [59] :

```

1) Tree1 = ROOT → Include → Contain
2) Tree2 = ROOT → Include → Incorporate
3) Least Common Subsumer(s) = argmax(depth(subsumer(Tree1,
    Tree2)))
4) Depth of Least Common Subsumer = depth(*ROOT*) = 1
5) Depth1 = min(depth({tree in T1 | tree contains LCS})) = 3
6) Depth2 = min(depth({tree in T2 | tree contains LCS})) = 3
7) Score = 2 * Depth of Least Common Subsumer / (Depth1 +
    Depth2) = 2 * 1 / (3 + 3) = 0.3333333333

```

Listing 4.2: Wu Palmer Sample Calculation[59]

Leacock-Chodorow Similarity (lch_similarity): This algorithm is very similar to the Wu-Palmer Algorithm except it calculates a negative logarithm of the path similarity. Let's give the same example comparing to "contain" with "incorporate":

```

1) Tree1 = ROOT -> <include> -> <contain>
2) Tree2 = ROOT -> <include> -> <incorporate>
3) Lowest Common Subsumer(s) = argmin(length(subsumer(Tree1,
    Tree2)))
4) Length(incorporate) = 1 and MaxDepth (v) = 14
5) Score = -log(length(Lowest Common Subsumer)) / (2 *
    max_depth(LCS.pos)) = -log( 1 / (2 * 14)) = 3.332204510175204
    > lch_threshold (equal to 2.15)

```

Listing 4.3: Leacock-Chodorow Sample Calculation[59]

Question Classification: A question answering system regardless of domain type needs a question classification algorithm to choose the best answer matching. It is a part of question processing that can parse the question input and assign into the correct labels. Machine learning methods can define derivation of an expected answer. Logistic Regression and Support Vector Machine methods have been used for question classification. The semantic question answering does not use a multi-layer perceptron, which is one of the deep learning methods, thereby taking a long time to train data.

There are a couple of methods based on logistic regression and support vector machine that a question classification can use.

4.4 Chapter Discussion

//Why did we use turtle and sparql

//data preparation discussion with regarding research questions

//natural language processing element discussion. Refer to literature review-

//How can we differentiate our solution

//Answer research questions

5 Practical Implementation

The practical implementation was realized with ASP.NET Core, Angular 6, Flask Micro framework and a template language named Ninja. This frameworks and template language will be enlightened in the following sections. As ASP.NET Core and Flask were using for back-end development, Angular 6 was used for front-end development. General Information about the architectural design of the web-based software can be found in Appendix B.1.

5.1 The RESTful Architectural Design in the Operator Assistant Web-Based Software

The REST stands for Representational State Transfer, which exposes a set of methods and endpoints in establishing decentralized applications where a client and a resource communicate with each other. An HTTP request method should encapsulate with an endpoint of RESTful implementation by ensuring stateless, cacheable and separation of concern regarding client/server communication. The RESTful interface is a basis for designing an architecture, which has components, may dispatch messages by HTTP methods in order to stay in contact with one component to another. Such methods indicate the desired action, and they can be GET, POST, PUT, and DELETE.

In Figure 5.1, each black arrow represents a RESTful communication utilizing HTTP methods. Alongside microservice architecture pattern has been used, the selection of architectural pattern varies from Service Oriented Architecture (SOA) according to the use of the representational state transfer (see Appendix B.1). In the practical implementation, there is no need to burden XML based negotiation protocol such as SOAP and complexity operations requiring addressing and content to be arrange over again. As shown in Figure 5.1, the web-based software rely on three basic components that deploy separately. For instance, blue and gray coloured components can be deployed separately and there is no dependency between them. One can decouple the orange and gray-coloured components, which are the OPC UA Web Component and the Semantic QA Component respectively, without changing any addressing, content or negotiation protocol. After adding a load balancer which can retrieve resources on behalf of a web user by balancing the requests, system performance and security might be boosted up. The Semantic QA interacts with a backup page, data cloud of eniLINK and natural language server

5 PRACTICAL IMPLEMENTATION

toolkits. The backup page has no login authentication, but it has JWT authentication module that can verify the private token of HTTP requests. If this component would have installed separately, it may harm to security. Nevertheless, one can add a login-authentication to that module or a transport layer security can be added such as HTTPS via the common load balancer.

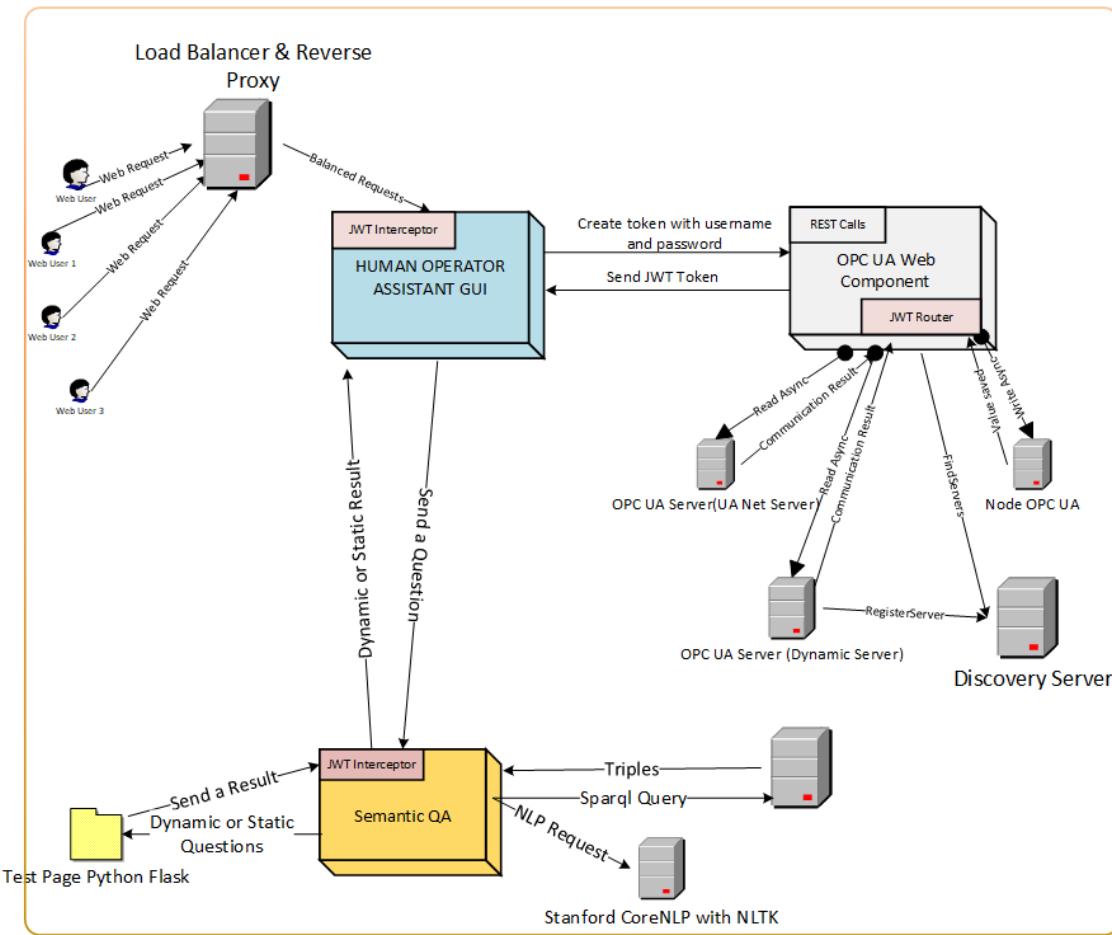


Figure 5.1: RESTful Architectural Design of the Web-Based Software

ASP.NET Core commences the API entry point with “API” keyword. All regardless sort of requests is being sent with “*api*” is a mandatory condition. For the first time, the body of a request should carry a username-password pairwise to initiate token-based authentication. HTTP method should be GET because this method has a feature called idempo-

tency. HTTP requests works the way of pipelining request to reduce transaction numbers between request and response. Although this feature increases the quality of request-response communication, the POST method may get in a danger, hence multiple requests of POST never been triggered in error conditions. The idempotency does not change the resource at multiple requests, namely, there is no difference aspect of resource between first and fifth requests. Caching can reduce redundant data transfer and round trip time through saving local storage instead storing into servers. The drawback of the caching is that response can be cached and displayed to unauthenticated user. Another drawback the data content may remain same whatever a request has been made by a user.

The OPC UA Web Component holds username-password pair in the way of embedding into the application. Hence, body of requests will take the username-password pair for authentication token as established in Listing 5.1.

```
[HttpGet("api/authenticate/") Body of Request {username, password}]
```

Listing 5.1: HTTP Get Request for token-based authentication

In particular, a node-id is a mandatory field in the request as below in Listing 5.2. “*serverconf*” parameter takes the ID of server which defined in “*appsettings.json*”. We have saved hard-coded server lists, but we can also obtain from a discovery service in the form of string array.

```
[HttpGet("api/serverconf/DataSetID/allnodes/{node_id}) Authentication Bearer  
{JWT}]
```

Listing 5.2: Http Get Request [7] [6]

The web-based software is capable of writing a value into a writable object. This is a limited feature against servers because most of the OPC UA Servers have some restrictions in order to protect from vulnerable attacks. However, simulated data in any object can allow writing in the structure for the testing purpose. The “*message*” can be a primitive or user defined type and it totally depend on writable attributes or value of a node. “*DataSetID*” is utilized for selecting hardcoded OPC UA Server and “*allnodes*” has been used for separation from “*subscribeNodes*”.

5 PRACTICAL IMPLEMENTATION

```
[HttpPost("api/serverconf/DataSetID/allnodes/{node_id}) {value:"message"}  
Authentication Bearer {JWT}]
```

Listing 5.3: Http Post Request [7] [6]

The Semantic QA does not use “*api*” routing to provide an easy integration to the OPC UA Web Component. As depicted in Listing 5.4 and Listing 5.5, separation of message type is important to realize while considering the Semantic Question Answering. Having multiple HTTP request is mandatory for the semantic QA, due to lack of semantic separation between queries about which a request is concerning streaming data or static data.

```
[HttpGet("/integratedstaticmessage/{question} Authentication Bearer {JWT}]
```

Listing 5.4: Question Answering Static Message HTTP Get Method

```
[HttpGet("/integrateddynamicmessage/{question} Authentication Bearer {JWT}]
```

Listing 5.5: Question Answering Dynamic Message HTTP Get Method

The subscription request consists of a monitor id to create a polling mechanism. As mentioned in Listing 5.6, subscription requests need to set monitor id arrays. To update “*monitor_id*”, the web-based application utilizes a POST request. A sample preview can be found under Appendix B.15.

```
[HttpPost("/api/serverconf/DataSetID/subscribeNodes/monitor_id) Authentication  
Bearer {JWT}]
```

Listing 5.6: HTTP Post Request for Monitoring Node

HTTP Request can be repetitive methods triggering by a user. Taking into consideration an increasing amount of data in industrial networks on a daily basis, an architecture should comply with the load balancing and non-blocking input-output queues. A load balancer can balance workloads by using the predefined algorithms. In this work, we will use the round-robin algorithm and the least connection algorithm. The round robin choose a guidance for workloads by putting into queues according to predefined weight. The least connection considers the workload which takes different

amount of time and it states that the fewest loaded queue should be selected avoiding stall of the workloads.

5.2 Front-End Development

5.2.1 Overview

Angular 6: Angular 6 is an extension framework that develops a variety of properties of Angular JS. Libraries of Angular 2 are not suitable to work legacy usage, so Angular JS Framework cannot use any library from Angular 2. The main reason for the underlying framework has been written in TypeScript, not in the JavaScript library. Angular 2 has a new command line feature that extracts essential information from “*package.json*”. All sort of libraries is saved into “*package.json*” to detect discrepancies between versions of libraries. Angular 6 allows embedding dynamic bootstrapping features into a pure HTML Page. The biggest drawback of Angular 6 is that it is not backward compatible, and the differences between versions can be immense. While Angular JS follows the pattern of MVC, Angular 6 implements a component pattern. Besides, Angular 2 splits component by component to increase code reusability and achieve the object-oriented paradigm in script languages. Angular 6 is faster than AngularJS versions because Angular 2 uses different hierarchical dependency for each module. When a module updated by a developer, only regarding hierarchy updated so that the front-end framework can enhance the running and compilation performance. A developer can use an external asynchronous event library in AngularJS, but Angular 6 provides an internal library implementing an asynchronous feature.

Ember.Js: Ember.JS is a JavaScript MVC Framework that helps to organize large web applications. The structure of Ember.js is depended on micro-libraries [61]. MVC pattern fully complies with Ember.Js in terms of bindings, computed properties and automatically updated templates [61]. Bindings enable the change of a variable propagating to another variable. Calculated Properties and Automatically Updated Templates ensure the framework stay up to date with regarding data source of Ember.js. One of the major advantages of Ember.js is Ember Data Library which stores all values of a process employing caching into an In-Browser Store [61]. Ember.js supports all end-to-end testing tool such as Karma and Mocha. Testability is an important step to develop bug-free codes so that one can state Ember.js has a variety of compliance with test tools.

The primary purpose of Ember.JS is to support a Single Page Application; thus it has no architectural layer for server-side rendering. Server-Side Rendering is an old transfer technology for HTML Websites and brings a significant overhead in case of minor changes. In addition, Server Side Rendering works with static sites that need to load the entire structure of web pages. However, the initial page loading time of Server Side Rendering is shorter than Client Side Rendering does. Ember.Js is fully backward compatible that means one can use a function from an old version in a new version.

React: The React Framework serves the purpose as a full-viewer of a front-end library. React is primarily concerned with the view aspect of UI and it is not suitable to use as a framework or library in a large-scale application [62]. React does not enlarged support for the following necessities: HTTP Calls, Routing, Dependency Injection are robust components when implementing a Web Service, so React cannot be taken into account a good solution for full-scale web service but the viewer. This could be a big drawback while comparing with the AngularJS framework. React has been posited that front-end developers can leverage its features to create the part of a viewer in MVC. React follow does slightly follow the MVC Pattern.

MeteorJS: MeteorJS is an open source project which has built on a stack of MongoDB, Node.js, Angular, and Express.js have consistent client-server applications, reactive modules, and rapid prototyping [63]. The underlying structure is based on Node.js and its virtual box named Google V8 Engine. The underlying mechanism of MeteorJS detects the changes of the object and automatically set the results before a developer made. Angular2 and React have observables to ensure this set of property.

VueJS: VueJS is a frontend framework that has a similar grammatical structure to ReactJS and AngularJS. Templates are one of the powerful features that used by VueJS. Through templates, the VueJS provides data bindings. Templates support two-way data binding, that means when you changed an input, VueJS will update the corresponding element. After combining VueJS element with HTML, every element of VueJS will be reactive, which inputs are rendered immediately accordingly. VueJS is a component-based system that the abstraction mechanism of language works with components. Methods can be called in VueJS through cached memory. Thus, a cached method is not compiled in multiple calls so that a VueJS application can reduce the memory complexity of method calls.

One-way bindings only propagate the changes into one single direction. Two way data binding allows to implement data flows two directions.

One can see the main characteristics of front-end frameworks and the underlying script languages Appendix B.8 and Appendix B.4 respectively.

5.2.2 Implementation of Front-End Development

Angular 6 applications have component-based object-oriented script languages, DOM elements of components and component export file. Export file is responsible for denoting the files under the same component to called “*NgModule*” in Angular 2. Although there is more typing into several files about structure, it discriminates by modules, components, and services to orchestrate with a starter component called “*AppComponent*”. The “*AppComponent*” bootstraps modules that an Angular 2 project includes by providing “*Http Interceptors*”. “*Http Interceptors*” consists of request functions and response functions of an HTTP Request. Chaining interceptor is a most common technique in Angular 6 and it defines “*req*”, “*res*” and “*next*” handlers. “*req*” handlers used to send an HTTP Request with a particular method such as “GET” or “POST”. Asynchronous communication compels each request that has been sent need not to wait for a consecutive request. The Chaining handler named “*next*” then put the result of “*req*” and “*res*” handler into a queue.

In fact, individual HTTP requests need a publish/subscribe pattern to ensure event handler works in a single thread with multiple events. Observables are used to ensure publish/subscribe pattern in Angular 6. Subscribers put HTTP Request into the event handler to send a request. This request value would be saved payload, event handler trigger and an error trigger. Whether any error had been occurred by server side, client application show result with “*HttpErrorResponse*” inside of constructor that holds error, particular headers, status, and URL information.

Components represent classes of Angular 6 that connects DOM elements such .html file in connecting with data binding. Each component has a selector and template URL to connect html files. Such selectors can be “*input-form*”, which is turn out to be a form file that an HTML file resides. Components delegate data access for Angular 6 services. “*@Injection*” annotation provides services in the context with dependency injection. Dependency Injection decreases the dependency of code organization if a class function uses a method in another class. “*@Injector*” annotation enables extending application and makes the unit test possible in inter-class dependency.

Document Object Model (DOM) manages markup languages and stylesheet document which organizes the display format of markup languages. DOM elements and Component object-oriented script files are in a robust two-way data binding relationship. Data binding is an essential feature in Angular 6, so a user can browse between nodes in OPC UA Server interactively. Updated JSON data should be viewed on a graphical user interface. In the case of data-intensive and nested JSON data, a loop should handle data to put into a data table. “ngFor” and “ngIf” are the fundamental statement used in Angular 6. A loop and a control statement can traverse through JSON format in a data table in which an OPC UA Server send instantly after triggering a click event by a user.

Having a JSON Web Token interceptor in front-end applications, back-end modules (QA back-end and OPC UA back-end) ensures data integrity with access and refresh token. JSON Web Token may provide a stateless and stateful connection across all incoming and outgoing requests. In our case, the front-end application takes a stateless JWT information by an Authentication Bearer. Front-end and back-end applications have a single secret key to verify the origin of the data with a symmetric key named “*HS256*”. This type of authentication has less overhead for both front-end and back-end side.

To improve our architectural design, a second front-end application for the Flask framework has been designed separately to increase modularity and separation of concerns. In this way, microservice framework (Appendix B.1) would be possible to future use at a different facility of a smart factory. Question Answering leverages a template-based front-end application, but the DOM elements are generated in the same back-end structure. However, multi-threaded Flask application strongly applies integrity of HTTP requests without delay and communication failure. Another advantage is that there is no third party package requirement for data-binding and static evaluation of question answering. Consequently, single page application can handle data the way of dynamic binding and single time loading of DOM elements from different back-end application that have used different programming language.

5.3 Back-End Development

5.3.1 Overview

The following section is a brief description of the back-end development process in terms of a framework that has used in experimental development of OPC UA, Information

Model Mapper for Semantic Data and Semantic Question Answering. All of these development cycles are examined with a comparison between frameworks, languages, libraries and toolkits. Regarding OPC UA Web Application, frameworks, languages, and software toolkits are taken into account. As far as Semantic Question Answering and Address

ASP.Net Framework (Active Server Pages .NET): ASP .NET Framework one of the oldest framework used by developers to implement web applications. The oldest framework named as ASP.NET Web Forms. ASP.NET Web Forms was strongly dependent on the Windows Operating System because of Internet Information Service (IIS). This server used for deploying web applications that can work only within the Windows Operating System. Moreover, this framework was limiting changes due to an internal file of the Windows Operating System as known as Web.dll [64]. Web Forms evolved to ASP.NET MVC Framework to comply with Model-View-Controller design pattern.

ASP.NET Core: Besides the continuous improvement of this technology, Microsoft Company decides to scale this framework Unix-based architecture. Therefore, the name of the technology changed as ASP.NET Core, which brings to the developer worlds lightweight features. One of the most prominent features of ASP.NET Core is the routing framework to control Rest API calls. When an HTTP call arrives, it should be parsed as a schema and host path. Schema path decides which protocol used an underlying structure to deploy a call. An aspect of the query string to understand specific element, the host part contains path and query structure to discriminate an HTTP call from each other. ASP.NET Core mainly used for a production environment because of the immature step of development like ASP.NET Framework (Think about). To deploy a web application rapidly, ASP.NET Core is a better choice thanks to its lightweight functions, the code size of the virtual machine, open source code, and interoperability with Unix-based operating systems. Moreover, ASP.NET Core has legacy support with ASP.NET MVC and Web Forms so that the framework can extend internal functions with legacy projects. **//KESTREL or IIS**

Node.js: Node.js is a framework based on JavaScript language that leverages a virtual machine developed by Google Inc.⁶ Node.js is an event-driven server-side development framework. By leveraging a virtual machine named V8 Engine, the framework sends an

⁶ https://www.w3schools.com/nodejs/nodejs_intro.asp

event signal to the virtual machine rather than communicating an operating system itself. Node.js has a broader support for multiple operating systems because the virtual machine has been compiled for multiple targets. Not only the framework is compatible with a client-side script language, but also it can integrate callback functions with low-level compiled language such as C++ or C. This paradigm had named as Native Call in the software world that is very useful when a function result returned from a programming language managed by a virtual machine into a native language. A finite state machine implemented by C++ creates asynchronous callbacks until a garbage collection eliminates the objects of callbacks.

Java Spring Framework: Java Spring Framework is the closest architecture to ASP.NET Core in terms of package management, virtual machine based garbage collection, routing and dependency injection. It has an object mapper such as Entity Framework in ASP.NET Core that ability to connect with databases mapping object into database objects. By supporting modular development, the code can be split into modules and it is getting easier to handle with code size when a project source code's volume (size) increased. Spring Framework work with Plain Java Objects.

Flask Micro-framework: Flask is a micro-framework using by many software developers benefits of rapid prototyping. For the reason that we need to develop a rapid-prototyped solution, Flask helps developers in many aspects. The Flask is working with many versions of Python 2.x and 3.x. There is no complicated routing mechanism and utterly compatible with microservice development. Annotators are bounded but compact, which is making the learning curve of the framework higher. A Flask Application can ensure JWT Authentication and other security policies with external libraries. In the research phase, we faced that the biggest problem of Flask is a non-asynchronous structure. One of our proposal to remedy the problem is using a non-blocking input-output queue with an asynchronous task queue. A non-blocking input-output queue can create an internal load balancer by balancing all requests into a queue before it reached to the application. It is far more than making a routine asynchronous. When an asynchronous queue works, it selects a message broker to connect a non-blocking input-output queue. This overall system creates an internal “message-broker system”. To change simply version from 2.x to 3.x would be the second solution. Because Flask application does not have the “async” keyword that makes the routines asynchronous call.

Django Framework: Django Web Framework is a loosely coupled, high-level Python Web framework along with supporting Model View Controller pattern. It leverages the language property of Python allowing indented programming and implicit data types. Underlying pattern is a bit different from MVC because the view part could present views through templates. The framework operates across multiple tiers such as Business Logic, Application, and Presentation Tiers. Django uses a particular template to fetch iterative values from template engines, and it is believed that the templates shorten the code complexity of Front-End. Django has a package manager called “pip” to organize libraries in a virtually separated folder. Main issues about Django are not occurring from the architecture of the framework, and rather it is associated with versions of Python 2.x and 3.x creates discrepancies among libraries. Routing Mechanism provided by regular expressions with precedence rules. When an URL is matched, all other requests are dropped in accordance with precedence rules. Django can consolidate URL Patterns by including a URL one into another. In this way, developers can easily manage the URLs and HTTP Requests within a single base URL entry point.

In the same way, the latency is a parameter indicating the duration of the delay within consecutive queries. In this test, each request is processed to fetch a single row from a database, and the data is serialized as JSON [65]. Hence, low latency gives better performance for back-end frameworks.

5.3.2 Authentication and Discovery of the Web-Based Software

A web-based software must have compliance with an authentication standard anyhow. Principally, there are two kinds of authentication, which can be observed in the web-based software, are certificate-based authentication, and token-based authentication for end-to-end security. End-to-end security emphasize that the communication medium may be affected by eavesdropping attack on the communication endpoints, but if one encrypts the data with a private key, then there will be no issue to obtain data by a malicious person without a private key.

On the communication level, a web application can handle the authentication issue as well. OPC UA Client and Servers can employ three kinds of secure connection, which are HTTPS, TLS or WebSockets. For instance, Hypertext Transfer Protocol Secure (HTTPS) or Transport Layer Security (TLS) provides encrypted data exchange on the transport level between receiver and sender, but it does not ensure end-to-end security.

Those two communication models every time need to send initialization parameters such as ACK, SYN to establish a new connection between client and server. Unlike HTTPs and TLS, the WebSocket protocol does not need to establish initialization parameters once it did. WebSocket prominently has more robust design than HTTPs and TLS, but support standardization among OPC UA Servers is very low. For example, [Cavaliere, Salafia & Scroppi 2018] [7] [6] [35] has an implementation is ensuring the WebSocket connection with a particular technology designed by Windows ⁷. This approach solves the standardization problem for the WebSocket protocol barely has solved, therefore, the web-based software only support connecting via binary connection with HTTP with certificate sharing.

OPC UA Protocol introduces a certificate-based authentication before establishing a session. The operator assistant web-based software provides a JSON web Token (JWT) Authentication. With JWT Authentication, a token is created by the back-end application of web-based software to send to a client side after a client performed an HTTP Request to an endpoint. A client or front-end application should send this token with every request that he wants to authorize while a process is executing referring token ID, name of issuer and issued time as Unix epoch date. The last part is verification signature, which consists of information regarding header, payload or encoded message.

The carrier system called Authentication Bearer, which is carrying out a body of a request in HTTP Protocol. A user types username and a password to get access a token to fetch data from a web-based system. After initiated username-password pair check, “*JwtSecurityTokenHandler*” creates a handler of a token and “*SecurityTokenDescriptor*” launches a description of a token. “*SecurityTokenDescriptor*” defines expiration date and type of credentials such as Aes128, HmacSha384 (Symmetric Encryption) or “*RsaSha256Signature*” (Asymmetric Encryption). In our case, the practical implementation of a symmetric key has with “*Hmac Sha1 256*” Bit Cipher.

A compact way to provide security is to implement an authentication method within low-level protocol area. For catering to protocol-level security, OPC UA Client employs a certificate-based authentication through an X509 Certificate to a certification routing as illustrated in Figure 5.2Figure 5.2: Authentication System in the Practical Implementation . This routing system prepares an authentication initiative to decrypt parameters of a certificate. At the same time, an OPC UA Client sends a secret message through Open Secure Channel initiated by a Session and message-certificate item pairs verified

⁷ <https://github.com/SignalR/SignalR>

with an asymmetric signature. “*What if the web-based application has been integrated into different division of a smart factory, how would the central part manage certificates?*”. This question cause to be unavoidable fact implementing a “*Local Discovery Server*” and “*Global Discovery Server*” as discussed previously in the Section 3.2.4.

A couple of examples for the discovery services have been given with main features in Appendix B.14.

JWT Authentication module of the OPC UA component covers the Question Answering component. Instead of creating multiple secret key, single key in a main component and confirming the data integrity of the main component is a more significant development practice, yet, it is a bad practice for the microservice architectural thinking in terms of separation of components. However, the authentication service of the general design can be separated for each components easily and this does not harm the design principle, and performance.

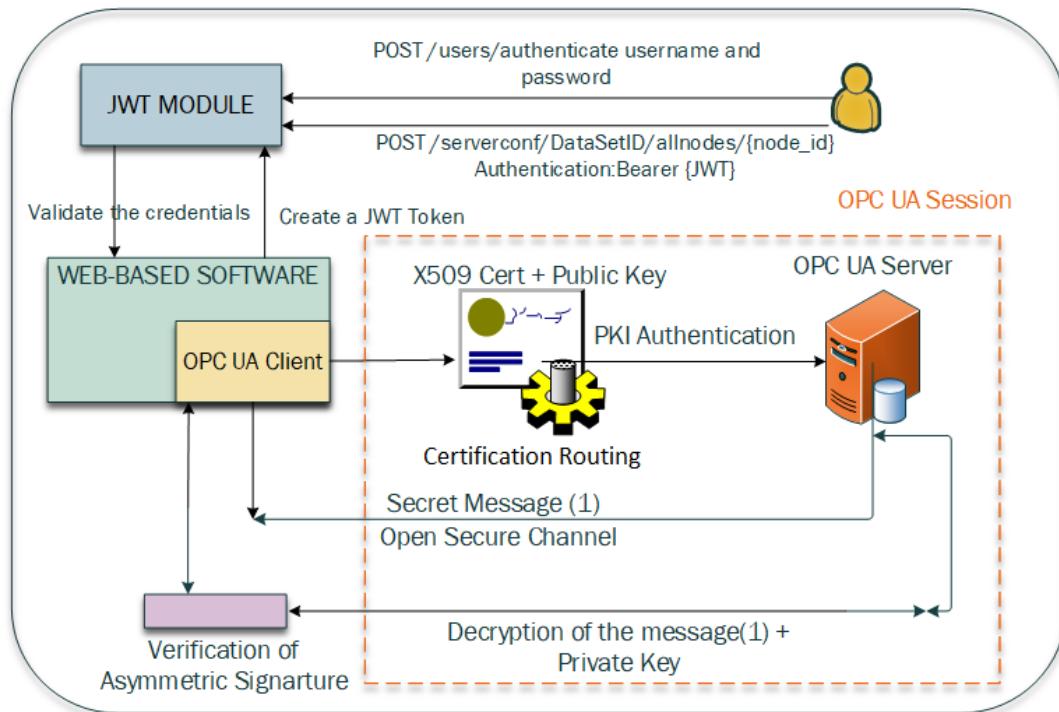


Figure 5.2: Authentication System in the Practical Implementation

Lastly, JWT Authentication is not the only method that a system can utilize through an authentication concept. Kerberos and OAuth can be thought instead of JWT Token for

the authentication concept. In common, all three authentications employ issued tokens, but there are different from each other slightly. JWT is an unloading authentication, which means that an authentication system interrogates tokens in incoming requests. A request that consists of JWT Token is not requiring extra configuration against endpoints. However, Kerberos Authentication needs a key distribution service and an authentication server. Initially, an incoming request sent to an Authentication Server to authenticate the request by user name and password pair. If a symmetric key used for authentication, the key distribution server should handle all requests to exchange keys instead of public-key infrastructure. This could lead to point of failure and security vulnerabilities. Another drawback about Kerberos, time limitation of a ticket does not allow time-to-live value such as JWT and OAuth2 and this time restriction must synchronize with clocks between servers.

//Close up with certificate authentication

5.3.3 Connection Establishment of OPC UA

OPC UA Connection has been achieved with .NET OPC UA Standard SDK [66]. A session need to control OPC UA client and server sides with asynchronous communication. Asynchronous communication can show benefits in the case of multiple OPC UA Server connections. “*CoreClientUtils*” provides an interface function named “*SelectEndpoint*”. A web-based software that has OPC UA client functionality can get endpoints with an array from a discovery service or hard-coded listed way. Before initiating a session, built-in certificate should match between clients and servers. A server behaves incoming client requests as untrusted communication at the initial stage. So manually accepting of certificates could be a necessity intervention for the sake of compact development. To eliminate this kind of intervention, a local discovery server may provide satisfactory approach. All OPC UA servers in the same network trusts the discovery services as they started to send notification on which they started. An unknown client can connect to a local discovery without initiating a session; hence, there would be no issue in trusting certificate between clients and servers.

Connection status between servers and clients might have polling by reading particular node value that shows server status or monitoring node concept. Monitoring node can support effective polling smaller time interval than reading node value. However, this approach creates extra overhead for client-side. Reasonable way to handle this using a communication stack function of a SDK that polls to server status under a structured data type such as "*DataValue*".

Reading, Writing, Browsing nodes, value or attributes uses particular asynchronous Session notification. This notification put the session information into mutex (thread-safe locking method) to ensure another request would not use the same session. Servers can fail incoming request under the circumstance that server thinks security or connection parameters of clients are not complied with default parameters. In this case, a client need not to restore a session, they should open new sessions to connect servers. Nevertheless, this could lead the client into degrading performance without shared session pools and alteration the timeout values.

5.3.4 Data Navigation and Polling through OPC UA in the Web-Based Software

As being a part of the OPC UA Information Model and Address Space, the Data Navigation is used for reading and writing request in OPC UA. The serialization and type management realization has been taken the implementation of previous studies have contributed to the thesis that has published by [6] [7] and Free OPCUA [50] in terms of user-defined structured data serialization and JSON Data Serialization. OPC UA utilizes tree-based hierarchical architecture to traverse among nodes with their references. Folders organize Address Space and they can abstract objects into Information Model. Complex type as predefined structures should comprise primitive type that can be reachable by OPC UA Client.

OPC UA Protocol defines its own data structures that break up into two main sections: Built-In and Structured Data Structure. When an OPC UA Client demands that is navigating through OPC UA Server, he should start from a root folder that consists of a root node. By sending a browse request to root note id that is equal for all standard OPC UA Server is "*namespace index= 0*" and "*node identifier=85*", OPC UA Client reach the terminal nodes of the tree structure in OPC UA Server. Generally, leaf nodes give standard information about folder, object and variables and continuous simulated data saved into the leaf nodes of OPC UA Server. OPC UA supports various data types up until top-level nodes in terms of object-oriented network design. The practical implementation

follows another way around like there is no hard-coded root node and namespace pairs for lucidity. An aspect of the web-based software, each user can send separate requests by creating a new session.

Navigation between nodes should consist of attributes and references as mandatory. Principally, a node attribute comprises the Browse Name and Node Id. “*BrowseName*” and “*Guid*” parameters as defined in OPC UA Protocol show initial names of nodes. Browse Names are matched onto Values and Datatypes. Browse Name and Display Name similar to each other except that Browse Name represents itself with a namespace index. Practical implementation exhibits a node id – namespace pairs, browse name, type of data and type of reference. However, an aggregation server can be created as a cumulative address space. All connected servers to the aggregation server create their own address space in order to do so. In this case, the aggregation server defines multiple root nodes with namespace and a GUID. The practical application can be extended to show multiple roots, but the scope does not cover this point. Root Object has three items, which are Objects, Types, and Views. Views is a restricted address space created by an OPC UA Server. Restricted spaces have different definitions for Views that limits Nodes and References. Views are more useful when used a cumulative address space in opposition to a single address space because multiple address space should discriminate more dynamic and static data from multiple OPC UA Servers.

The idea behind of serialization is converting OPC UA Server built-in types into JSON Schemas or Values. By way of JSON serialization, a web-based service can use the information through HTTP Request Payload to communicate between front-end and back-end architectures. Even though the features of built-in data such as description, binary schema, field type etc. saved as an XML schema in OPC UA Protocol, the XML format is not suitable for neither OPC UA Web-based software communication among modules nor a semantic question answering. By this means, an overhead of conversion of syntactic XML is not a problem while developing a web-based software or a semantic question answering. The approach comprises the Structured Data Type and Built-In Data Type which is converting into a JSON Format through serialization to send a proper response from OPC UA Servers [7] [35]. To navigate between Structure Types used an XPath navigation includes over 200 built-in functions for string values, numeric values, Booleans and node manipulations [67]. Most built-in types are encoded in XML Schema of OPC UA Standard Definition. A Client holds application configurations, data types of nodes, and security information with certificates as encoded in XML Schema. Due to being a

structured data type, information parsing are relatively easy with labels within an XML Schema.

As for subscription service of the web-based application, this study handles a subscription request with a minimum sampling time interval of integer monitoring node id to register either a variable, an attribute, a node or an event. The Web-based software does not specify any minimum sampling time interval. In the practical implementation, one can prepare a packaged notification message with required monitored node id. The limitation of the monitored node is not all OPC UA Servers provide monitorable structure for variables or events. This restricts to follow all changes for manufacturing device and the only solution could be redesigning OPC UA Server in order to subscript changes within a particular time interval. Subscription connects to an existed session to prevent creating a redundant number of sessions. A lived session can be controlled from a common pool that has all session's identification numbers with their generic configuration.

After subscribing a request, one can fill up the subscription with a monitored node id. At this stage, the system should assign a sampling interval rate. The rate implements a cyclic rate that the server can sample data from real items. Sampling rate selection could be problematic in the implementation. A user or an internal application can make the selection. For instance, monitored part of the application select 1000 milliseconds sampling interval, which is the most common interval rate selected by OPC UA Servers. If the 1000 milliseconds has been selected although a server does not support the rate, the server assigns the most applicable rate in order to apply a sampling rate. The selection process may be different server by server. Regardless of a sampling rate below than a particular value or upper than that, server creates a subscription to insert a monitored node into the subscription. User can take an exception from protocol stack regarding mismatching interval rate, either way a subscription is produced. Another finding is that the underlying structure of subscription mechanism of OPC UA Servers is not thread safe. That means a client implementation should be aware while multiple monitoring nodes could create a delay for results. **Consequently,**

5.3.5 Implementation of the Semantic Question Answering

Our approach is heuristic-based syntactic analysis and each question should follow the steps of query formulation phase follow the flowchart as shown in Figure 5.3 with this approach. An input should clean unnecessary characters with stop word removal and

tokenization functions. All of the elements illustrated in Figure 5-3 has represented in different classes in an application. The stemming clears the prefixes to compare the synonym of the word. The stemming makes clear prefixes and suffixes of verbs. The lemmatization turns given verb into a concise form by eradicating English tenses. Unlike an open-domain question answering, we could not implement a rule-based approach. For instance, [Unger Et al. 2012] proposed a template-based approach gives precise answers with a predefined set of a statement such as <noun, property, object> into template SPARQL. Theoretically, it is one of the closest solution for our case to achieve proper results. In practice, template-based approach decrease easy of use concept with regards to usability of a human operator. But it can decrease the steps Figure 5.3 which it may increase speed performance of the semantic question answering.

As illustrated in Figure 5.3, each color represents different process of the QA. While the parsing is a syntactic finder as shown in green, due to specialization process, the Question Classification and Query Formulation steps are shown in a different color as orange and red. The rest of the process is normalization process, namely it clears unnecessary parts of input.

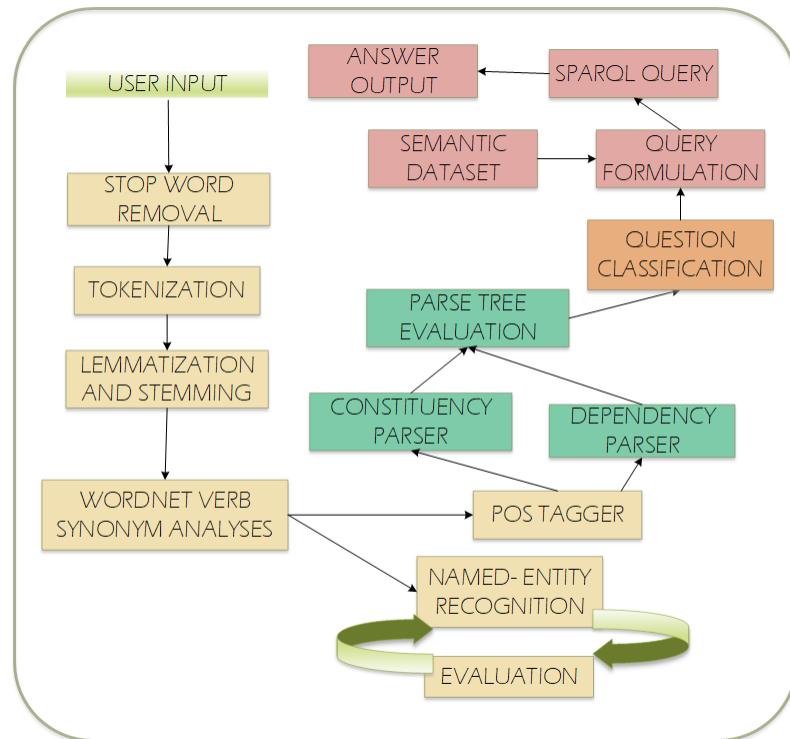


Figure 5.3: The Flowchart of the Semantic Question Answering

If a question answering system gets a large scale or identified data of data, the system can score answers of questions to indicate in a better way. In practice, answer selection, which provides the most convenient answer out of collected answer set, is not suitable for the semantic question answering in restricted-domain according to our experiment. Data scarcity does not only affect implementing a machine-learning algorithm but also makes a question answering based on information retrieval harder. That means a semantic question answering that exploits restricted source should focus on an in-depth parsing approach. After implementing the semantic question answering, a question classification used for eliminating answer types rather than scoring the answers.

The semantic question answering employed a limited linked data represented as Turtle RDF for static queries. The statements i.e. “What does linkedfactory contain” or “Please give me all of its members” were asked to the question answering. One of our findings is that an upper or lower case of question can produce different results from constituency parser unless we turned into a lower case of them.

Algorithm 2 Query Formulation

```

1: function QUERY FORMULATION(naturalinput)
2:   query  $\leftarrow$  QueryWithPrefixes
3:   r  $\leftarrow$  constituent.parse.tree
4:   indirectdependency  $\leftarrow$  dependency.parse.tree
5:   while nodes  $\neq$  leafs.terminal do            $\triangleright$  Until leaf nodes(Terminals)
6:     verbs  $\leftarrow$  PARSER(nodes)
7:     nouns  $\leftarrow$  PARSER(nodes)
8:     similarityflag  $\leftarrow$  WORDLATENANALYSIS(verbs)
9:     if StaticInformation is True then
10:      indirectdependencyFlag  $\leftarrow$  DEPENDENCYPARSER(nodes)
11:      if similarityflag and IndirectDependency is true then
12:        object  $\leftarrow$  nouns
13:        predicate  $\leftarrow$  verbs
14:        query  $+=$  object + predicate + ?subject
15:      else
16:        subject  $\leftarrow$  nouns
17:        predicate  $\leftarrow$  verbs
18:        query  $+=$  ?object + predicate + subject
19:      if DynamicInformation is True then
20:        predicate  $\leftarrow$  PARSER(nodes)
21:        object  $\leftarrow$  PARSER(nodes)
22:        similarityflag  $\leftarrow$  SENTENCESIMILARITY(input)
23:        query  $+=$  object + predicate + ?subject
24:   return query

```

Figure 5.4: Query Formulation Algorithm: The source code can be found under the link
<https://github.com/zointblackbriar/QuestionAnswering>

Figure 5.3 explains all process as a black box and the flowchart does not give the detail of query formulation. Subsequently, the Query Formulation has to map syntactic elements onto SPARQL query patterns. A question can change predicate, object and subject order according to a human operator required. In this case, the dependency parser detects

//Talk about shallow parsing deep parsing constituent syntactic parser

For streaming query, the algorithm as shown in Figure 5.4 evaluates the queries within two main sections. While static queries are sent against local linked data endpoint, dynamic queries need an API to get through federated services. A human operator must do this separation when he or she asked. Numerical keywords such as “value”, “average”, “minimum” or “maximum” are key points of a semantical separation between

static queries or dynamic queries. Regex-based rules are used to discriminate keywords as mentioned above.

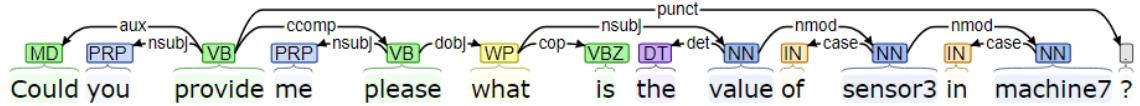


Figure 5.5: Dependency parser

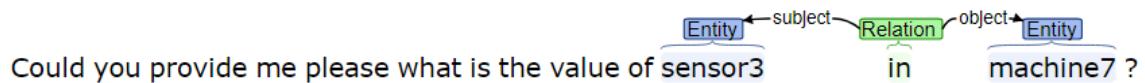


Figure 5.6: Named-entity recognition

To implement the question classification we have realized the algorithm ideas (logistic regression with Limited , Linear SVC) and test datasets (TREC data and manually generated dataset) of the research [68][69] in the implementation part. The common point of datasets have questions and labelled class of these questions. Questions are grouped with coarse-grained labels, which are “Abbreviation”, “Entity”, “Description”, “Human”, “Location”, and “Numeric”. Manually generated dataset that we have trained with Logistic Regression which comprises of “what”, “quantity”, “who”, “unknown”, “affirmation” and “why” labels. The question classification utilizes the tokenization of bi-gram language modelling.

//Short explain of question classification

6 Experimental Results

In Section 6, we will explain the test methods and environments to create a critical mind about operator assistant web-based software. We evaluate the OPC UA Client feature of the web-based software with some parameters, e.g., viability of unit testing, mock testing, and load testing. We separately evaluated the Semantic Question Answering with the precision of answers and usability of the question answering. Our main theoretical motivation is to provide an assessment of performance for the web architecture. OPC UA Client is hard to test with mock testing because creation a session at the protocol level for each request could cause failure on overall web-based software. A load testing would use for testing functions of the web-based software, e.g., opening a session, sending a request, serialization of objects, and closing a session. A timeout value of testing tool and OPC UA Client should overlap; otherwise, results of performance or load testing can give us wrong results in terms of failed requests. The Semantic QA needs various metrics as defined in Section 6.2. The list of meanings has been listed in Chapter 7.2 Materials and Methods. RESTful API requests are equally important for testing as performance testing. Randomly selected requests with JWT authentication and their results are shown in Listing 7-1. In the evaluation phase chiefly three questions I will be asked for

- 1) What is the accuracy of the question answering according to generated questions?
- 2) What is the performance and degree of functionality of the assistance web-based application?
- 3) How well the web-based application react in case of balancing workloads.

6.1 Test Methods and Environment

The Test Environment covers functionality, performance, and accuracy of the web-based application afore-mentioned questions in Section 6. Functionality and performance evaluation is based on the architecture behind the web-based application. A load test has been planned and used to test web-based software against OPC UA performance to do so. Accuracy and relevant evaluation metrics will evaluate the Semantic Question Answering. Manually generated and streaming test set has been used to enlighten predefined metrics for QA. OPC UA performance testing has still some delicacy in which they

would use mock and unit testing with OPC UA Client. Therefore, we decided to test with HTTP Request all the way up from front-end application and back-end application in order to establish a deduction of hypothesis 1 (1.4). The load test is broken into two phases that each part has evaluated a different number of HTTP Requests,

At the first stage, we have the following conditions to evaluate the performance of the web-based application with a single request without load balancing. This test will evaluate the core components of web-based software. Then we change the test environment by adding a load balancer. Load Test takes a single request and multiple requests to test for each test case. In light of multiple requests, requests can be dispatched into one OPC UA Server, but each request are aiming to distinct node in the server.

We realized tests that we have formed in the Test Methods three times over again.

System Property: Intel® Core™ i7-2720QM CPU @2.20Ghz 16 GB Ram x64 Windows Operating System

Test Software: West Wind Web Surge (Build 1.10), Node OPC UA Server (Master v0.5.5 Branch), and Nginx Load Balancer (v 1.5.4 – 27 Aug 2013 – Non-Commercial Version)

Test Constraints: Nginx in Windows OS supports the test environment up to 500 users.

Test Methods: We send a single HTTP Get Request to Node OPC UA Server with a load balancer and without load-balanced single core back-end application within 30 seconds to calculate total throughput (total number of requests), throughput per second (request per second), and average response time.

In the second test, we send multiple HTTP Get Methods to Node OPC UA Server with a load balancer and without load-balanced single core back-end application, (Kestrel embedded web server). Additionally, we send multiple requests to the Question Answering with 10 users within 30 seconds to ensure the H 1).

Test Parameters: We set an HTTP header that uses the front-end application of Angular 6 as a proxy with HTTP Referrer. This referrer takes incoming requests as if it is coming from the front-end application.

Test Condition: All components including front-end application and the Semantic Question Answering are up and run. Embedded firewalls are offline.

6 EXPERIMENTAL RESULTS

Expected Deviation: +/- 30 of total requests, -/+ 5 Throughput (Request per second), -/+ 200 milliseconds (average request time)

For the functional tests, we defined six HTTP Request that evaluates different functionality. These tests have been defined according to security and modularity. We grouped the functional tests as insecure authentication in front-end, unappropriated HTTP method, and false token authentication.

For question answering test, we have asked 50 questions to the QA. Questions are manually generated question that can be found in Appendix A.1. The data sets of the question answering system is examined in Section 6.2.

//Why did not we test with POST requests

//Cache could make faster and high amount of requests since we tested GET Requests.

The reason why the web-based application imposes GET requests mostly is about idempotency and caching problems in HTTP requests. Furthermore, as being sent a POST requests, idempotency can affect the general results and POST request does not contains JSON serialization. So we will see more request but less valuable.

PUT and DELETE requests are out of scope due to data integrity.

6.2 Question Answering Data Sets and Evaluation Metrics

//how the data was gathered and what it is intended to represent

//what the gathered data looks like

//how it should be interpreted.

//Data sets are given in Appendix

A well-founded evaluation metrics that other studies in the literature review have used. [Nguyen, Kosseim 2004] has evaluated their study Okapi Formulation often called Okapi weighting uses answer scoring. However, knowledge-based question answering does not use an answer selection model because data sets are limited to obtain enough answers. Hence, we will apply machine learning metrics which called Precision, Recall, F1-Score, and Accuracy. We have generated data by Dynamic Server (Appendix B.5)

which has a size of 2.216 KB and 39374 lines. Other static data is originated by eniLINK which has a size of 19 KB and 73 lines.

[Diekema, Yilmazel, Liddy 2004] offered five different categories which are listed in Table 6.2. The Answer Return Rate is a performance metric to evaluate an answer to a question regarding round-trip time. The time metric shows the duration that it takes a request to go from a starting point to an end-point back-and-forth. The Querying Style is about a user may ask factoid or keyword-based questions. The size and the coverage shows data size with lines and relative size and domain of data that we have obtained respectively. Up-to-dateness indicates whether the Semantic QA supports update statements or not. Lastly, the Query Formulation Assistance has features of the Semantic QA relevant to ease of use.

True Positive stand for an answer of a question that has correctly labelled in case it is predicted positive. False Positive is meaning that an answer that has incorrectly labelled false in case it is predicted true. True negatives says that the answer is going to label as false but the actual case is already false. Unlike true negative, false negative states that an answer of a question would be false but we have actually predicted as true.

Precision = True positives / (True positives + False Positives)

Recall = True positives / (True positives + False Negatives)

F1-Score = $2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall})$

Accuracy of the Model = $(\text{True Positive} + \text{True Negative}) / (\text{True Positive} + \text{False Negative} + \text{False Positive} + \text{True Negative})$

6.3 Results

As illustrated in Table 6.1, we have six mock testing to test the functionality of the REST Interface. Results are evaluating security and integrity of the message, for example, ID numbered 1, 2, 4, and 5 is evaluating front-end application routing and error message according to the authentication mechanism. ID numbered 3 tested multiple writing requests that is coming from front-end application and it shows that the test is passed.

6 EXPERIMENTAL RESULTS

HTTP Call	Test	Expected Output	Result	ID
[HttpGet("api/authenticate")]	Send the request without JWT.	HTTP Not Found 404. Action S	OK	1
[HttpGet("api/serverconf/DataSetID/allnodes/{node_id}")]	Send the request without JWT.	OK 200. Reroute to the login page	OK	2
[HttpPost("api/serverconf/DataSetID/allnodes/{node_id}") {value:"message"}]	Send multiple consecutive request	OK 200. HTTP Response with JSON	OK	3
[HttpGet("/integratedstaticmessage/{question}")]	Send the request without JWT	404 Not Found. Reroute to the login page	OK	4
[HttpGet("/api/serverconf/DataSetID/subscribeNodes/{monitor_id}")]	Send the request with the wrong JWT	404 Not Found. Reroute to the login page	OK	5
[HttpGet("/integrateddynamicmessage/{question}")]	Send a PUT Request with right JWT:	405 Method Not allowed.	OK	6
[HttpPost("api/serverconf/DataSetID/allnodes/{node_id}") {value:"message"} Authentication Bearer {JWT}]	Send a POST Request to the restricted address space in an OPC UA Server	500 Internal Error	OK	7

Table 6.1: Functionality Test Results

We are collaborating authentication part; question answering and OPC UA part of the web-based application to show results of HTTP Request that was undertaken by the practical implementation. Test call numbered six respond with a “Method Not Allowed” message which is a denial request coming from the web-based application. Totally, we have six test cases of the functionality. ID 1 test case is the initial authentication after taking username-password pairwise. All other test cases must be contingent upon the creation of a token that in case authentication was granted. If the Angular 6 could not

find the right token that the ASP.NET Core back-end framework generated, the front-end application should reroute to login page. ID 4 and 5 test cases confirms the validity of JWT tokens in terms of the front-end application.

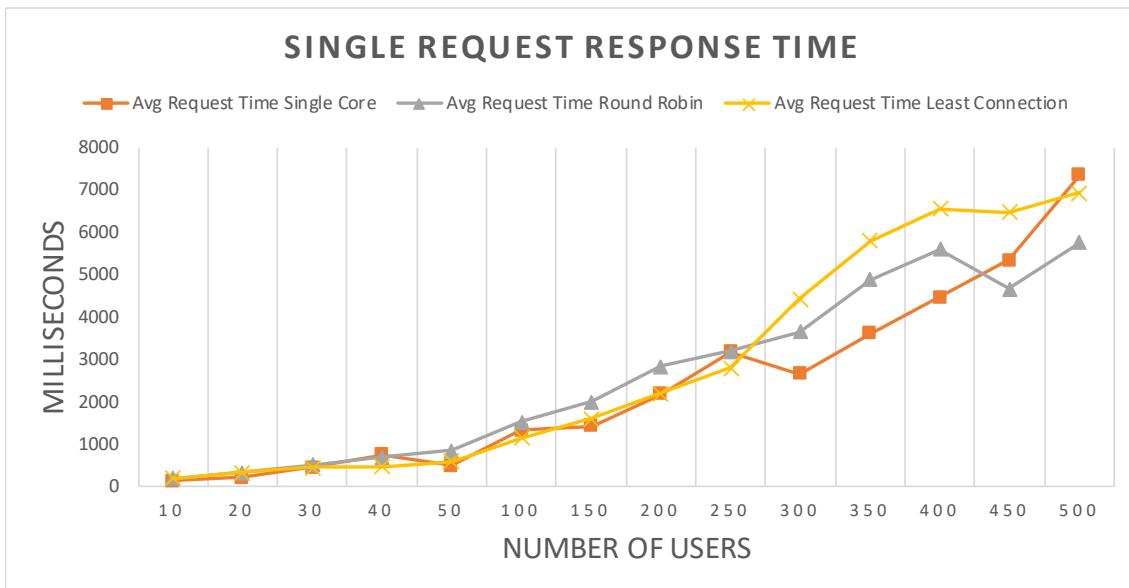


Figure 6.1: Single REST Request Response Time

As illustrated in Figure 6.1, single core, the round-robin algorithm and the least connection inclined to rise below 8000 milliseconds. At the first 50 users, average request time does not change in the midst of single core, round-robin and least connection distinctly.

As the round-robin was showing higher milliseconds until it reaches 250 users, single core and least connection were depicting similar inclination. Up to 250 users, architectural design thinking algorithms closely tend to incline at the same rate. Interestingly, there is an overlapping point at 250 users for each method. The lower milliseconds of the average requests, the faster an end user of web-based software can obtain results. Single core of the back-end application dramatically increased approximately 2500 milliseconds from 450 users to 500 users. Each 50 iterations of sessions, which users create a thread simultaneously per session.

6 EXPERIMENTAL RESULTS

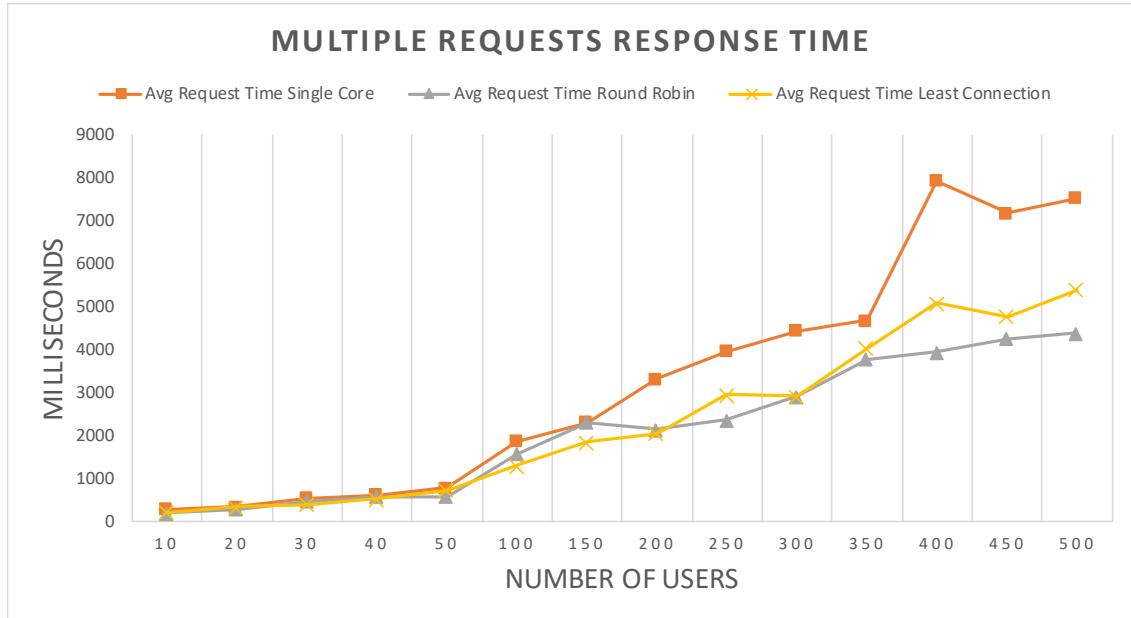


Figure 6.2: Multiple REST Request Response Time

In the Figure 6.2, Single Core average request time increased slope more than shown in Figure 6.1. Under multiple requests, there is a jumping point while passing through from 350 to 400 users. This caused a remarkable increment of the single, which is that, other algorithms of load balancing never reacted similarly. Round robin reached the lowest average request time. The round robin and least connection has performed operation within less duration that the single core does. While each 50 increment of number of users affect a smooth step-up for the single core on the regular basis, the least connection made a zigzag movement starting from 200 users.

The round robin displays better performance than the least connection and the single core does, which a query has been responded at most 4000 milliseconds under the round robin algorithm.

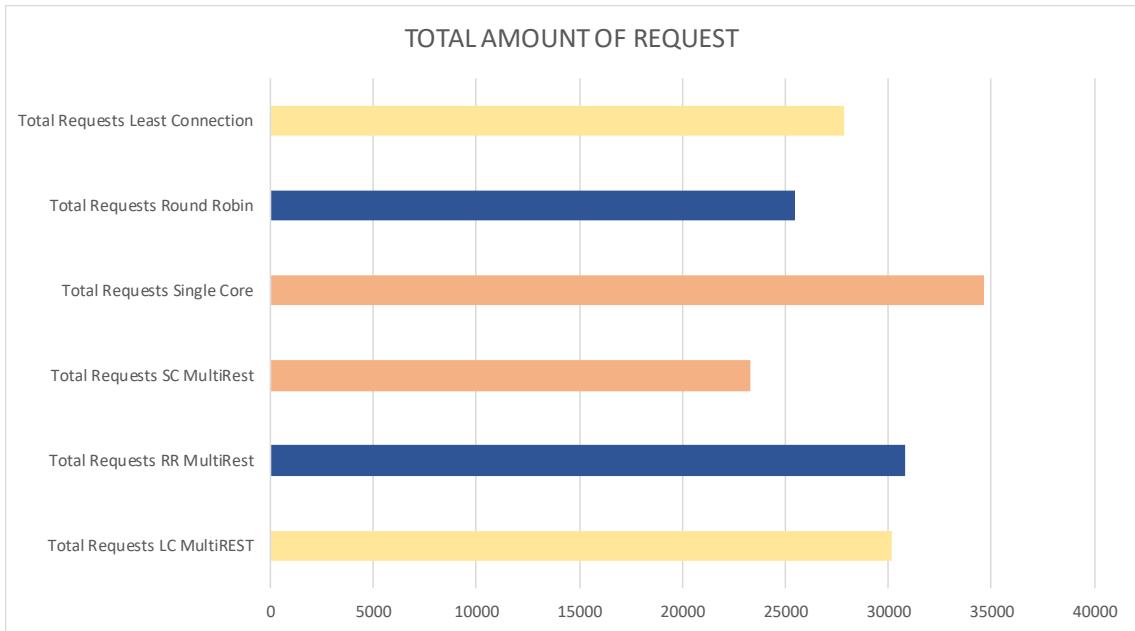


Figure 6.3: Total Amount of REST Requests

The total amount of request evaluation predicted alteration of the architecture concluding in the sense of HTTP communication. In Figure 6.3, “SC MultiRest”, “RR MultiRest” and “LC MultiREST” stand for “Single Core Multiple Requests”, “Round Robin Multiple Requests”, and “Least Connection Multiple Requests” respectively.

The round-robin has increased 5000 Requests, and the least-connection algorithm increased the requests approximately 3000 under the load balancing. However, a total number of requests dramatically decreased about 10000 requests upon altering the architecture from non-load balancing to load balancing. Briefly, Round robin increases %18 at total throughput, Least Connection increases %10.71 at total throughput, the Single Core loses %29.34 at multiple HTTP requests decisively.

6 EXPERIMENTAL RESULTS

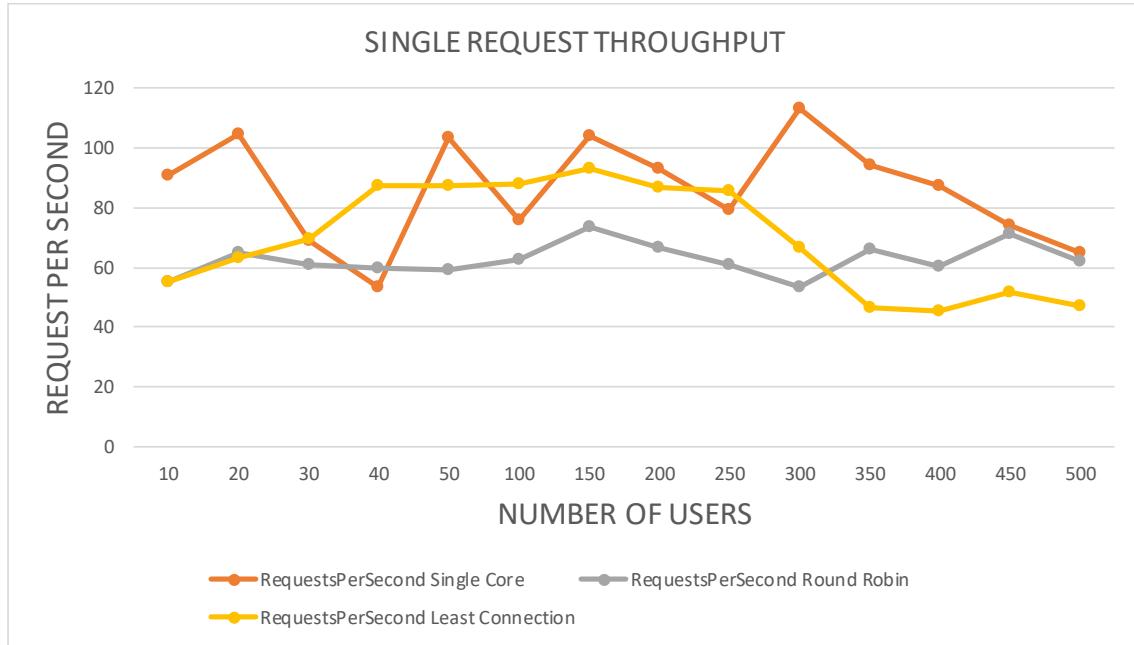


Figure 6.4: Single REST Throughput

As illustrated in Figure 6.4, the single core appears to be some fluctuations while creating throughput. Such fluctuations does not exist in load-balancing algorithms. After 350 users has been testing, the least-connection shows lower throughput than the round-robin algorithm did. The fluctuations of single core has been spotted four times over the 100 request per second. Neither round-robin algorithm nor least connection could not achieve to create requests per second above than 100. In practice, the round robin and the single core reach same amount of throughput at 500 users, but the least connection reached 10 request lower than the rest of methods made. Despite, the round-robin has performed balanced path with single HTTP request, the least connection decreased the throughput per second after it passed 250 users. Apparently, the single core reduces averagely 40 requests between 300 users and 500 users. Consequently, the throughput per second is higher for the single core and round robin than the least connection.

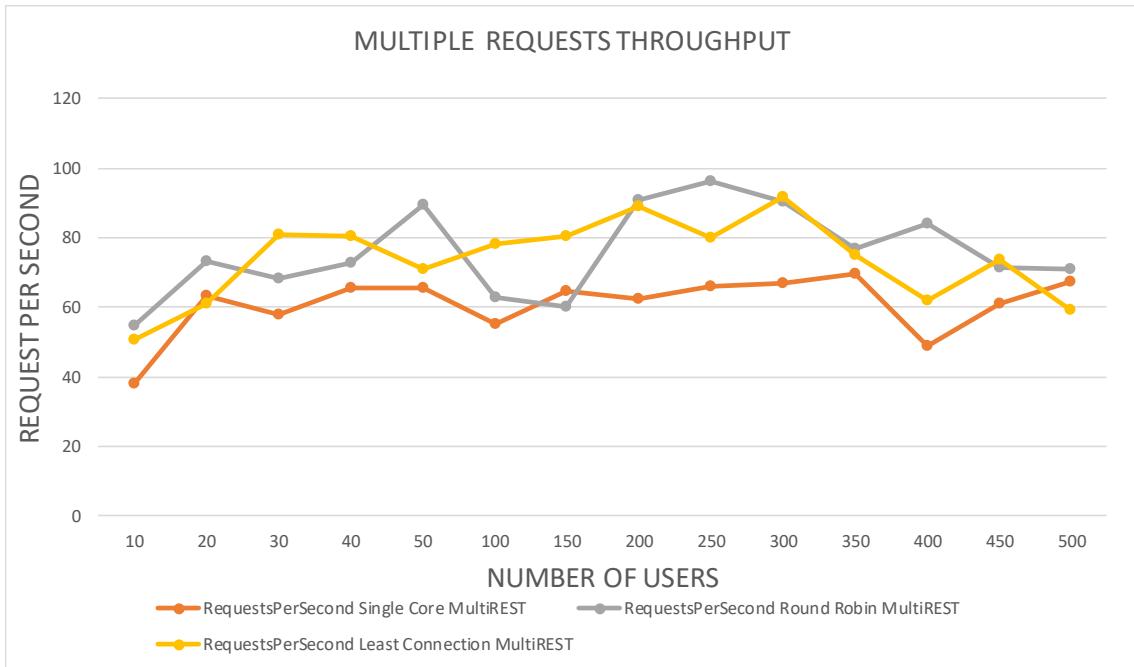


Figure 6.5: Multiple REST Throughput

As shown in Figure 6.5, the round robin generates the multiple highest throughput at 50, 100, 450 users. The single core generally goes on the lower amount of throughput than the least connection and the round robin. The lowest throughput for the single core is at 400 users, but the web-based application reached a peak of throughput approximately more than 45 requests as shown in Figure 6.4. The web application can respond %21.42 higher throughput under multiple requests with the round robin. In Figure 6.5, the single core generated the requests that remain more stable than Figure 6.4.

No tests that have evaluated the performance of OPC UA Component of the application took a failed response. This is because we have equalize the timeout value of OPC UA Client (60000 ms) to request timeout.

The following information is concerning results for the Semantic Question Answering component of the web-based software.

The Semantic QA has a performance result called “*Answer Return Rate*”, which is test under the same back end framework. Consecutive query and initial query has a disparity

6 EXPERIMENTAL RESULTS

aspect of round-trip time with 27 seconds. However, there is no disparity for static queries regardless of consecutive or initial queries. Theoretically, there is no difference between static query application and generated data application. In practice, having bootstrapped loading the word vectors and universal dependencies, the practical part could avoid repetitive loading for each query.

Evaluation Parameters	Properties
Answer Return Rate	QA against generated data from OPC UA – 23.25 seconds average QA against static query from RDF file of eniLINK – 18.92 seconds average QA against dynamic query from streaming data – 17.48 seconds QA against Open-Domain Questions – 20.55 seconds
Querying Style	Keyword-Based Search and Semantic Search
Coverage	eniLINK data, linkedfactory streaming data
Size	Static data relatively small size (39447 lines, 2235 KB) Continuous data relatively large size (Depends on time-series value)
Up-to-dateness	No SPARQL Update or Inference statements
Query Formulation Assistance	Voice Input Recognition, Spell Checker

Table 6.2: Evaluation parameters of the Semantic Question Answering

The open domain question answering is not our scope, but we compared with a result of a pattern-based open domain question answering. In practice, this type of question answering should have been faster than our approach in Section 5.3.5. Obviously, it shows similar round-trip time as much as the semantic question answering does. Data size is relatively changeable between a static search and a dynamic search. Dynamic search exploits streaming data generated by a time series and a key-value database.

A user can search with keywords or semantic (sentence or question) words the results within the question-answering module. . The semantic question answering does not al-

low updating to a triple because changes of triples in Turtle Source can create a vulnerability. Last but not least, query assistance has been provided by the semantic question system to improve the search quality and ease of use.

Question Answering	True Positive	False Negative	False Positive	Precision	Recall	F1	The Accuracy of the Model
Total Questions	34	13	3	%94.44	%72.34	%81.92	%68

Table 6.3: Total answers from Semantic Question Answering

We have obtained the F1 Score, Precision, Recall and Accuracy of the Model. We classified answers of the questions as four labelled, which is named multi-class classification. We do not have any true negative value; hence, we did not put the true negative column into the table. We have 34 correctly answered question out of 50 questions. As we can see, we obtained high precision value, but the Recall value is more important for evaluation. In this case, we are going to miss the correct answers of %72.34 questions. If we have 100 questions that we should answer them, we will miss the correct answer at most %27.56 of the total correct answers. The precision value %94.44 indicates that we have built classifier with high precision. F1-Score is being perceived a balance factor between the precision and the recall. Since the precision and the recall value is high, we may state that the precision proves less trade-off among the recall and the precision.

Consequently,

7 Discussion

In the beginning, research questions and hypothesis were defined in order to elucidate our research goals. We will summarise the key findings according to results of experimental results. Due to synthesized idea, there is no particular research for human operator solution combining with web science, artificial intelligence and industrial network protocol. However, we will discuss the results relate to the state-of-the-art studies that we have examined in Section 2. Conclusively, we will elaborate implication of practical results about how this affects our original hypothesis with Section 7.3, and then we conclude the research in Section 8.

7.1 Summarization of the Key Findings

The first finding is related to the OPC UA Web Component, it has been standardized on communication protocols and service sets. Nevertheless, there is no common architectural implementation aspect of between clients and servers. For instance, some OPC UA Servers have strict protection regarding session establishment, size of view.

The second finding was refreshable token realizing in short amount of time can degrade the throughput per second and total throughput, despite it can increase the strength of security. One can decide this trade-off according to requirements in a division of smart factory.

The third finding was that performance of the human operator assistant software can change with balancing workloads. Every manufacturing division has a different necessity as per the number of users and their functional expectations. Total amount of requests and throughput per second would be higher with load balancing round-robin and least-connection algorithm against multiple HTTP requests.

The fourth finding was that the stateless communication and stateful communication is still hard to integrate. Even a wrongly set timeout value by OPC UA Client module can increase the network failure, which leads to bottleneck of the web-based application. Moreover, establishing a distinct session for each request can cause overflow to an OPC UA Server if there is no protection against session workload overflow.

7.1 SUMMARIZATION OF THE KEY FINDINGS

The fifth finding was that the semantic QA slightly depend on the type of domain and data set. If the domain and data sets changed in different division in a smart factory, a property semantic extraction algorithm would be a necessity to map these properties onto <subject, predicate, object> triples.

The sixth finding was that the semantic QA is not suitable for mission-critical systems. Fetching a data within a constrained time is possible with template-based, pattern-based or modelled machine learning methods. Even if one may implemented these solutions above, high computational power is a must to find out complex morphological elements of semantic query and extracting a model from complex data sets.

During the performance tests, the web-based application displayed a couple of unexpected results. Firstly, the web-based software created fluctuations as illustrated in Figure 6.4. These fluctuations starts from the beginning of user request until 300 users have send requests. Unexpectedly, the web-based software did not show any fluctuation pattern under multiple HTTP requests in Figure 6.5. Thinking as a whole, Figure 6.5 should have displayed kind of pattern because of heavy loading, but not in Figure 6.4. As a result, asynchronous calls can send all attached requests in a single session. When a single query employed to different port each

The second unexpected result was occurred in Figure 6.3. Normally, higher number of HTTP requests should generate HTTP response in a particular duration of time. In Figure 6.3, the single instance that has been worked by OPC UA Web Component generated fewer results after the system applying multiple HTTP requests. A potential result is that the OPC UA Web Component is not successful enough to balance workload while applying multiple HTTP requests.

The third unexpected result was the Semantic QA throws an error regarding multiple threaded environment within a load testing. If a common load balancer used for distinct components, the risk of single point of failure may be increased. The problem is here that the Semantic QA works as synchronously, unlike the OPC UA Web Component. Throughout the entire application run and up, no error case has not been observed, but the web-application generates relatively small amount of total number of requests according to the experimental results.

7.2 Challenges in the Practical Implementation

The first challenge was that the polyglot implementation, which is realizing each component with a different programming language, is a challenging activity. Although the requirements shaped the level of polyglotism, replying to all requirements with sample programming languages can reduce the separation of concern.

The second challenge was related to representing the multiple data from multiple monitoring node in a stateless protocol is hard. Instead of using two-way data binding of Angular 6, an aggregation server can be implemented that holds monitoring node in a different hierarchy and the web-based application can connect it.

The third challenge was that the heuristic based syntactic parsing can not produce a model just like a supervised machine learning method does. It should start from the beginning as shown in Figure 5.3 and all process takes considerable amount of time which shows lower degree under a performance test. In order to solve this problem, template based questions and SPARQL can be implemented. But this solution reduces the expandability and easy of use. Other solution could be that one can use a machine learning algorithm instead following the stages of Figure 5.3, but the data size, coverage and quality must allow complex machine learning algorithm.

The fourth challenge was that the creation of manually generated questions and data sets from Dynamic Server quasi OPC UA Server. Due to the challenge of natural language processing, manually generated questions can not be complex sentence or statements that have paragraphs. Likewise, data sets are difficult to obtain as classified data and it could have some non-uniform data sections that sampled from the majority class such as machine error or node id information. This blocked the use of machine learning methods which could be more efficient and boosted application. To solve this issue, a feature selection method can help detect clear boundary of each predefined class. Another solution is examining not only with accuracy, but also F1 score, recall, and precision altogether.

The fifth challenge was about testing regarding end-to-end functionality is hard to form. While testing the application, each component must be thought as a distinct test case since the nature of application complied with the microservice architecture. Another issue about the testing of OPC UA Client feature is not easy to implement with mocking and unit testing. The useful approach is testing with HTTP Request to observe all action

such as Service Establishment, Secure Connection Creation, Data Serialization and Session Response.

7.3 Assessment of Research Questions and Hypotheses

Existing algorithms and design principles in the previous researches can be applied to the operator assistant web-based software. RQ 1) mentioned the modularity and viability of OPC UA Services. At the protocol level, [Grüner, Pfrommer & Palm 2015, 2016] stated the use of stateless transport layer protocol instead of OPC UA could reduce communication effort between clients and servers in case of eliminating session establishment phase.

RQ 2)

RQ 7 requires a detailed analysis

8 Conclusion

8.1 Summary

8.2 Future Works

- 1) Future Research: Where to go from here (can include where NOT to go, if your research demonstrated that a particular approach or avenue was not useful)

The devices that connected to OPC UA are growing day by day and requirements are expanded according to new requirements of smart factories. Experts need more tools that can take natural input by giving a scientific output. In order to provide this, linked data sources should be more capable of representing internal devices, actuators, and sensors. Such HMI devices connected to a remote domain of a smart factory need a more complex discovery service like Global Discovery Service. A Global Discovery Service can handle certificate management between local and remote domain through a certificate distribution. So each device can authenticate through a global discovery service and OPC UA Clients can easily find endpoints of OPC UA Servers. OPC UA Servers can change data inside and enlarge with a hierarchical structure of a smart factory. To serialize such changes requires a continuous serialization process; however, a continuous SPARQL language can detect the triples on updated roots of OPC UA Servers. Many different serializations of OPC UA static data has been left for further development due to architectural difficulties. KVIN Service could be turned into a continuous SPARQL endpoint instead of key-value storage.

//Named Entity Recognition can customize for company needs.

//Aggregated Server for general requirements for Smart Factory and for monitoring nodes.

//Abbreviation Checker can be handled

//Linked Data Efficiency will be improved. Especially, we can add more properties and corresponding explanations through the linkedfactory web page.

//A microservice architecture can be planned

//Reason induction system can be added if more data would be provided

//A machine learning system based on deep learning can be implemented if we get more data that generated by Fraunhofer IWU.

//Spell Checker has been implemented

//Authorization is a higher-level representation so that one can implement a role-based authentication after authenticating a system. These roles can be broken into administrative and user roles. A system can assign different rights to these roles in order to provide a system's security or integrity.

//In a particular range of time, a user send a real time query against KVIN Service.

//Non Blocking IO Queue (e.g. Rabbitmq can be added to balance load)

//Sparql endpoint of KVIN should comply with the param
http://domain.com:10080/sparql/endpoint&{SPARQL Query}

//Multiple federated query support with “SERVICE” statement can be extended with the new version. SPARQL GRAPH statement can be extended in accordance with Update

Bibliography

- [1] F. IWU, ‘eniLink’, 2015. [Online]. Available: <http://platform.enilink.net/>. [Accessed: 23-Nov-2018].
- [2] VisualWorks, ‘Cincom Smalltalk ObjectStudio OLE User’s Guide’, Cincinnati, Ohio, P40–3805–03 ©, 2010.
- [3] Unified Automation GmbH, ‘Introduction to Classic OPC’. [Online]. Available: http://documentation.unified-automation.com/uasdkhp/1.0.0/html/_l2_classic_opc.html. [Accessed: 05-Dec-2018].
- [4] B. Farnham and R. Barillère, ‘Migration From OPC-DA to OPC-UA’, *Proc. ICALEPCS2011*, 2011.
- [5] Unified Architecture, ‘OPC Technologies’, *OPC UA Foundation* . .
- [6] S. Cavalieri, M. G. Salafia, and M. S. Scroppi, ‘Integrating OPC UA with web technologies to enhance interoperability’, *Comput. Stand. Interfaces*, no. August 2017, 2018.
- [7] S. Cavalieri, D. Di Stefano, M. G. Salafia, and M. S. Scroppi, ‘A web-based platform for OPC UA integration in IIoT environment’, *IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA*, pp. 1–6, 2017.
- [8] OPCUAUniCT, ‘OPC UA Web Platform’, 2017. [Online]. Available: <https://github.com/OPCUAUniCT/OPCUAWebPlatformUniCT>. [Accessed: 05-Mar-2019].
- [9] T. Paronen, ‘A web-based monitoring system for the Industrial Internet’, 2015.
- [10] S. Gruner, J. Pfrommer, and F. Palm, ‘A RESTful extension of OPC UA’, *IEEE Int. Work. Fact. Commun. Syst. - Proceedings, WFCS*, vol. 2015–July, no. 01, 2015.
- [11] S. Grüner, J. Pfrommer, and F. Palm, ‘RESTful Industrial Communication with OPC UA’, *IEEE Trans. Ind. Informatics*, vol. 12, no. 5, pp. 1832–1841, 2016.
- [12] R. Schiekofer, A. Scholz, and M. Weyrich, ‘REST based OPC UA for the IIoT’, *IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA*, vol. 2018–Septe, pp. 274–281, 2018.
- [13] D. Mollá and J. L. Vicedo, ‘Question answering in restricted domains: An overview’, *Comput. Linguist.*, vol. 33, no. 1, pp. 41–61, 2007.

-
- [14] D. Mollá and J. L. Vicedo, ‘Special Section on Restricted-Domain Question Answering’, *Comput. Linguist.*, no. October 2006, 2007.
 - [15] S. C. Tirpude and A. S. Alvi, ‘Closed Domain Keyword based Question Answering System for Legal Documents of IPC Sections & Indian Laws’, no. 2, pp. 5299–5311, 2015.
 - [16] H. Chung *et al.*, ‘A Practical QA System in Restricted Domains’, *ACL 2004 Quest. Answering Restricted Domains*, pp. 39–45, 2004.
 - [17] D. Diefenbach, P. Maret, V. Lopez, and K. Singh, ‘Core Techniques of Question Answering Systems over Knowledge Bases: a Survey QALD-QA benchmarks View project PowerAqua View project Core Techniques of Question Answering Systems over Knowledge Bases: a Survey’, 2017.
 - [18] L. K. Doan-nguyen Hai, ‘The problem of Precision in Restricted-Domain Question-Answering. Some Proposed Methods of Improvement’, *Proc. ACL Work.*, pp. 8–15, 2004.
 - [19] F. F. Luz and M. Finger, ‘Semantic Parsing Natural Language into SPARQL : Improving Target Language Representation with Neural Attention’, vol. 1803.04329, 2018.
 - [20] S. Werner, D. Moldovan, M. Tatú, T. Erekhsinskaya, and M. Balakrishna, ‘Semantic question answering on big data’, pp. 1–6, 2016.
 - [21] A. Celikyilmaz, ‘A Semantic Question / Answering System using Topic Models’, *Text*, pp. 1–4, 2009.
 - [22] C. Unger, L. Bühlmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, and P. Cimiano, ‘Template-based question answering over RDF data’, *Proc. 21st Int. Conf. World Wide Web - WWW '12*, p. 639, 2012.
 - [23] S. Palaniappan, U. K. Sridevi, and J. Subburaj, ‘Ontology based Question Answering system using JSON-LD for Closed Domain’, vol. 119, no. 12, pp. 1969–1980, 2018.
 - [24] S. Ferré, ‘SQUALL: A controlled natural language for querying and updating RDF graphs’, *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7427 LNAL, pp. 11–25, 2012.
 - [25] A. R. Diekerma and E. D. Liddy, ‘Evaluation of restricted domain Question-Answering systems’, *Cent. Nat. Lang. Process.*, pp. 12–16, 2004.

-
- [26] B. Katti, C. Plociennik, and M. Schweitzer, ‘SemOPC-UA: Introducing Semantics to OPC-UA Application Specific Methods’, *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1230–1236, 2018.
 - [27] J. Pfrommer, S. Grüner, T. Goldschmidt, and D. Schulz, ‘A common core for information modeling in the Industrial Internet of Things’, *- Autom.*, vol. 64, no. 9, 2016.
 - [28] X. Su, H. Zhang, J. Riekki, A. Keränen, J. K. Nurminen, and L. Du, ‘Connecting IoT sensors to knowledge-based systems by transforming SenML to RDF’, *Procedia Comput. Sci.*, vol. 32, pp. 215–222, 2014.
 - [29] X. Wang, X. Zhang, and M. Li, ‘A survey on semantic sensor web: Sensor ontology, mapping and query’, *Int. J. u- e- Serv. Sci. Technol.*, vol. 8, no. 10, pp. 325–342, 2015.
 - [30] K. R. Llanes, M. A. Casanova, and N. M. Lemus, ‘From Sensor Data Streams to Linked Streaming Data : a survey of main approaches’, vol. 7, no. 2, pp. 130–140, 2016.
 - [31] J. J. Calbimonte, H. Jeung, O. Corcho, and K. Aberer, ‘Semantic Sensor Data Search in a Large-Scale Federated Sensor Network Semantic Sensor Networks 2011 Semantic Sensor Networks 2011’, *Semant. Sens. Networks*, no. January, pp. 14–29, 2011.
 - [32] D. Anicic and P. Fodor, ‘[epsparql] EP-SPARQL: a unified language for event processing and stream reasoning’, *Proc. 20th ...*, pp. 635–644, 2011.
 - [33] D. F. Barbieri, ‘C-SPARQL: SPARQL for Continuous Querying’, *Proc. 18th Int. Conf. WWW*, vol. 427, no. c, pp. 1061–1062, 2009.
 - [34] H. Hasemann and A. Kröller, ‘The Wiselib TupleStore: A Modular RDF Database for the Internet of Things’, *J. Phys. Soc. Japan*, Apr. 2018.
 - [35] Universita degli Studi di Catania, ‘OPC UA Web Platform’, 2017. [Online]. Available: <https://github.com/OPCUAUniCT/OPCUAWebPlatformUniCT>. [Accessed: 14-Jan-2019].
 - [36] T. D. Oesterreich and F. Teuteberg, ‘Understanding the implications of digitisation and automation in the context of Industry 4.0: A triangulation approach and elements of a research agenda for the construction industry’, *Comput. Ind.*, vol. 83, pp. 121–139, 2016.
 - [37] R. Margaret and D. Daniel, ‘Definition of Smart Factory’. [Online]. Available:

<https://searcherp.techtarget.com/definition/smart-factory>. [Accessed: 05-Dec-2018].

- [38] C. Team, 'What is the smart factory and its impact on manufacturing?', 13 June 2018. [Online]. Available: <https://ottomotors.com/blog/what-is-the-smart-factory-manufacturing>. [Accessed: 05-Dec-2018].
- [39] K. Thoben, S. Wiesner, and T. Wuest, "Industrie 4 . 0 " and Smart Manufacturing – A Review of Research Issues and Application Examples', vol. 11, no. 1, 2017.
- [40] D. Gorecky, M. Schmitt, M. Loskyll, and D. Zühlke, 'Human-machine-interaction in the industry 4.0 era', *Proc. - 2014 12th IEEE Int. Conf. Ind. Informatics, INDIN 2014*, pp. 289–294, 2014.
- [41] J. A. Holgado-terriza, 'Mobile Human Machine Interface based in OPC UA for the control of industrial processes . Mobile Human Machine Interface based in OPC UA for the', no. August, 2016.
- [42] O. P. C. Unified, A. Specification, and S. Release, 'OPC Unified Architecture Specification Part 4: Services', *Specification*, vol. Part 4, no. Release 1.04, 2017.
- [43] O. P. C. Unified, A. Specification, A. Space, and M. Release, 'OPC Unified Architecture Part 3: Address Space Model', *Specification*, vol. Part 3, no. Release 1.04, 2017.
- [44] Unified Automation GmbH, 'Address Space Concepts'. [Online]. Available: http://documentation.unified-automation.com/uasdkhp/1.0.0/html/_l2_ua_address_space_concepts.html. [Accessed: 07-Dec-2018].
- [45] Mariusz Postol PhD. Eng. (Project Manager), 'OPC UA Information Model Deployment', Wolczanska, Poland, 2016.
- [46] O. P. C. Unified, A. Specification, and I. M. Release, 'OPC Unified Architecture Specification Part 5: Information Model', *Specification*, vol. Part 5, no. Release 1.04, 2017.
- [47] D. Jurafsky and J. H. Martin, 'Speech and Language Processing', *Speech Lang. Process. An Introd. to Nat. Lang. Process. Comput. Linguist. Speech Recognit.*, vol. 21, pp. 0–934, 2009.
- [48] P. D. Leslie F. Sikos, 'Mastering Structured Data on the Semantic Web'.

- [49] B. DuCharme and Beijing, *Learning SPARQL Querying and Updating with SPARQL 1.1 Bob*, Second Edi. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'REILLY, 2013.
- [50] Pure Python OPC-UA Client and Server, 'Free OPC-UA Library'. [Online]. Available: <https://github.com/FreeOpcUa/python-opcua>. [Accessed: 22-Nov-2018].
- [51] TU Dresden, 'Plt-TUD'. [Online]. Available: https://github.com/plt-tud/opc_ua_xml_export_client. [Accessed: 22-Nov-2018].
- [52] F. Breitling, 'A standard transformation from XML to RDF via XSLT', *Astron. Nachrichten*, vol. 330, no. 7, pp. 755–760, 2009.
- [53] C. D. Manning, 'Foundations of Statistical Natural Language Processing - Christopher D. Manning', pp. 1–704, 2005.
- [54] A. Clark, C. Fox, and S. Lappin, *Computational Linguistics and Natural Language Processing*. West Sussex, United Kingdom: Wiley-Blackwell, 2010.
- [55] A. Taylor, M. Marcus, and B. Santorini, 'The Penn Treebank: An Overview'.
- [56] J. Perkins, D. Chopra, and N. Hardeniya, *Natural Language Processing : Python and NLTK*. 2016.
- [57] neo4j, 'The Jaccard Similarity Algorithm'. [Online]. Available: <https://neo4j.com/docs/graph-algorithms/current/algorithms/similarity-jaccard/>. [Accessed: 16-Jan-2019].
- [58] P. Christen, 'A Comparison of Personal Name Matching: Techniques and Practical Issues', *Sixth IEEE Int. Conf. Data Min. - Work.*, no. September, pp. 290–294, 2006.
- [59] Wordnet Similarity for Java, 'WS4J Demo'. [Online]. Available: <http://ws4jdemo.appspot.com/?mode=s&s1=Is+the+system+health+is+good%3F&s2=What+is+the+status+of+system%3F>. [Accessed: 17-Jan-2019].
- [60] T. Wei and H. Chang, 'Measuring Word Semantic Relatedness Using WordNet-Based Approach', *J. Comput.*, vol. 10, no. 4, pp. 252–259, 2015.
- [61] J. H. Skeie, *Ember.js in Action*. New York: Manning Publications Co., 2014.
- [62] M. Tielens Thomas, *React in Action*. Manning Publications Co., 2018.
- [63] S. Hochhaus and M. Shoebel, *Meteor In Action*. New York: Manning Publications Co., 2016.

-
- [64] A. Lock, *ASP.NET Core In Action*. New York: Manning Publications Co., 2018.
 - [65] TechEmpower, 'TechEmpower Framework Benchmarks'. [Online]. Available: <https://github.com/TechEmpower/FrameworkBenchmarks>. [Accessed: 23-Jan-2019].
 - [66] OPC Foundation, 'OPC Unified Architecture .NET Standard'. [Online]. Available: <https://github.com/OPCFoundation/UA-.NETStandard>. [Accessed: 13-Mar-2019].
 - [67] W3C, 'Xpath Tutorial'. [Online]. Available: https://www.w3schools.com/xml/xpath_intro.asp. [Accessed: 14-Jan-2019].
 - [68] S. Khare, 'Question Classification Implementation'. [Online]. Available: <https://github.com/swapk91/Question-Classification>. [Accessed: 26-Feb-2019].
 - [69] Shirish Kadam, 'NLP:Question Classification using Support Vector Machines', 2017. [Online]. Available: <https://shirishkadam.com/2017/07/03/nlp-question-classification-using-support-vector-machines-spacyscikit-learnpandas/>. [Accessed: 14-Mar-2019].
 - [70] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, 'The Stanford CoreNLP Natural Language Processing Toolkit', *Proc. 52nd Annu. Meet. Assoc. Comput. Linguist. Syst. Demonstr.*, pp. 55–60, 2014.
 - [71] C. Richardson, *Microservices Patterns with examples in Java*, First Edit. New York: Manning Publications Co.
 - [72] Arnon Rotem-Gal-Oz, *SOA Patterns*, First Edit. New York: Manning, 2012.
 - [73] OPC Foundation, 'OPC Unified Architecture Specification Part 14: PubSub Release', 2018.

Appendix A The Semantic Question Answering

A.1 Manually Developed Test Questions – Precision and Recall

Question	ID	Precision	Recall
What do linkedfactory, heatmeter, and e3fabrik incorporate exactly?	1	0	0
Provide me a combined result for IWU and e3sim	2	1.0	1.0
I want to know which one carries fofab?	3	1.0	1.0
There is a member named fofab. Please give me all of its members	4	1.0	1.0
I am a customer for this company. Could you tell me please what the value of sensor1 of machine1 is	5	0	0
Could you tell me please what is the current value of sensor2 in machine2?	6	1.0	1.0
What POWERMETER holds?	7	1.0	1.0
What does FOFAB incorporate?	8	1.0	1.0
What does machine5 HOLD?	9	1.0	1.0
What does gmx comprise?	10	1.0	1.0
What comprises karobau?	11	0	0
System health for sensor2 in machine6	12	1.0	1.0

Tell me the health of system for sensor2 in machine1	13	0.0	0.0
Could you browse generated data?	14	1.0	1.0
Give me all of the members of gmxspanen4	15	0.0	0.0
What holds coolingwater?	16	1.0	1.0
What is the hierarchical structure of fofab?	17	1.0	1.0
What contains IWU?	18	A	B
Could you give me the members in which contained by versuchsfeld?	19	1.0	1.0
Could you give me the members in which linkedfactory has?	20	A	B
What is the value of sensor1 in machine6?	21	1.0	1.0
What is minimum that we can calculate for sensor1 of machine1?	22	1.0	1.0
What is the value of maximum can be calculated by the sensor1 of machine1?	23	1.0	1.0
Could you tell me what the average for sensor3 in machine1 is?	24	1.0	1.0
I need to learn an average value for sensor5 in machine2	25	0.0	0.0
What is the average of sensor3 in machine3?	26	A	B

Could you get me the references of nodes?	27	A	B
Could you browse generated data?	28	A	B
Is the E3-Sim member of linkedfactory?	29	0.0	0.0
Could you take me all members of generated data?	30	A	B
Give me all registered node id	31	A	B
I need to learn parent node id in generated data	32	A	B
Could you give me parent node id in the file of generated data?	33	A	B
Give me all data blocks	34	A	B
Data blocks in generated OPC file	35	A	B
Give me the name of stations in generated data	36	A	B
All stations which are in generated data or new data	37	A	B
Please combined result of datablock, station	38	A	B
Who is Fofab?	39	0.0	0.0
Why can all nodes be browsed?	40	0.0	0.0

Table 0.1: Precision and Recall of Answers

A.2 Natural Language Understanding Libraries

TextBlob: TextBlob is a tool based NLTK to process a natural query without providing NLTK's function overhead. It was written in Python 2 but also compatible with the Python 3 version. It provides a simple API for diving into common tasks of Natural Language Processing such as part-of-speech tagging, sentiment analysis, classification etc.⁸

Stanford CoreNLP: One of the fastest and robust libraries for Natural Language Processing provided by Stanford University. The only drawback of this library is limited support for Python programming language. However, it has been solved this problem with Rest – Compatible Web service by sending external HTTP queries from Python programming language. Stanford CoreNLP works based on Java Virtual Machine so that it can be conceptualized as model-view-controller pattern. As shown in Figure 1.1, Stanford CoreNLP supports diversity of implementation that is a vital role for natural language processing. Stanford CoreNLP provides an API which annotation-based that suitable underlying models or resource are available for the different languages [70]. The main drawback of CoreNLP is that one needs to use other programming languages except for Java by wrapping up Java compiled packages to specific languages. This reduces supports of full-feature such as sentiment analysis, dcoref, regexner⁹.

Spacy: It is an open source library for Natural Language Processing which is written in Python and C-counterpart Cython¹⁰ It utilizes the convolutional neural model for tagging, parsing, and entity recognition to increase the precision of findings in natural language processing. When a request is sent to spacy, it calls a language pipeline, which it is brought into line tokenizer, tagger, parser, and named-entity recognition respectively.

AllenNLP: It is a scientific based NLP library compatible with Python. AllenNLP also provides a demo tool which has used in this work to demonstrate the development steps of NLP. AllenNLP has advanced features to use that not only industrial scale application but also scientific purpose tools such as coreference resolution, semantic role labeling, open information extraction or textual entailment.

⁸ <https://textblob.readthedocs.io/en/dev/>

⁹ <https://github.com/Lynten/stanford-corenlp>

¹⁰ <https://spacy.io/api/>

SyntaxNet: It is a library provided by Google that works with a deep neural network based on Tensorflow. The main purpose of this library is to serve as a syntactic parser. Moreover, this library focuses on dependency parsing more than constituency parser.

Natural Language Toolkit: NLTK is one of the fundamental languages that consists of many features such as tokenization, parsing, tagging published as an open source project.

Libraries	Advantages	Disadvantages
TextBlob	Low overhead while doing a natural language understanding	Only windows based service setup,
Stanford CoreNLP	Strong support for variety of languages	Central Point of Failure, Bottleneck if there is not enough maintenance
Spacy	Easy to use with web platform technologies	Dependency on Node Virtual Machine and npm package manager Pipelining creates repercuSSION in NLP Limited Architecture Support(64 bit OS)
AllenNLP	Large supported-features for Natural Language Processing	No support for windows
SyntaxNet	Purely Deep Learning Based Stack based dependency parsing	No backward compatible. No asynchronous support for the language version. Python 2.x

Table 0.2: NLP Toolkits Advantages and Drawbacks

A.3 KVIN Service Sample Query

Query

Model
<http://linkedfactory.iwu.fraunhofer.de/data/>

Query

```
select * where {
service <kvin:> {
<http://localhost:10080/linkedfactory/demofactory/machine1/sensor1>
<http://example.org/value> ?v . ?v <kvin:limit> 1 ; <kvin:value> ?value
}
}
```

The SPARQL query.

Submit

Figure 0.1: Enilink Sample SPARQL Query

A.4 KVIN Service Result of Appendix A.4 with a Key-Value Pair

Result

v	value
:node1cvr8o4kfx2005	2.142857142857143

Figure 0.2: A result from a continuous data

A.5 Serialized Streaming Data into Linked Data

```

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <http://example.org/data/values.csv#>.

<http://linkedfactory.iwu.fraunhofer.de/linkedfactory/values.csv#row=1>
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory#time> "2018-09-
28T06:49:16.9230000+00:00"^^xsd:dateTime;
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory#value>
8.142857142857142.
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory/values.csv#row=10>
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory#time> "2018-09-
28T06:49:43.9260000+00:00"^^xsd:dateTime;
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory#value>
8.166666666666666.
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory/values.csv#row=100>
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory#time> "2018-09-
28T06:54:13.9650000+00:00"^^xsd:dateTime;
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory#value>
8.166666666666666.
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory/values.csv#row=1000
>
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory#time> "2018-09-
28T07:39:14.3010000+00:00"^^xsd:dateTime;
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory#value>
4.166666666666667.
<http://linkedfactory.iwu.fraunhofer.de/linkedfactory/values.csv#row=101>

```

Listing 0.1: Generated RDF from Real-Time Data Source

A.6 KVIN Streaming Data SPARQL Service

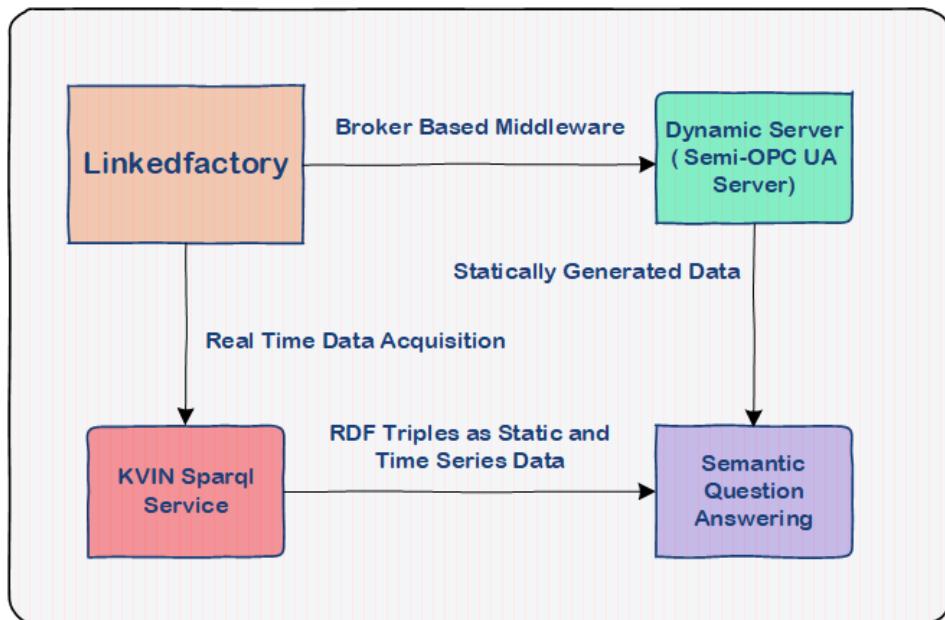


Figure 0.3: KVIN Service Relationships with Components

A.7 eniLINK Prefixes

```
@prefix : <enilink:model:users#> .  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix owl: <http://www.w3.org/2002/07/owl#> .  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

Listing 0.2: Enilink Sample Prefixes

A.8 Deep Parsing-Shallow Parsing

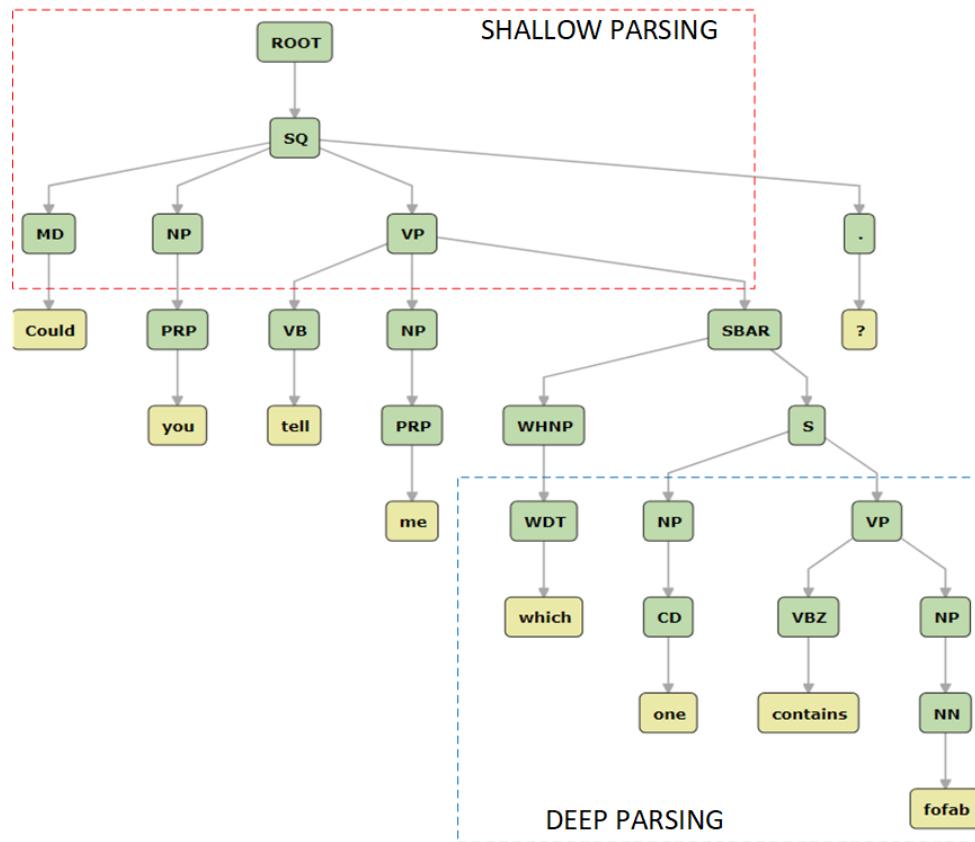


Figure 0.4: Constituent parse tree for factoid questions

A.9 Generated Answer from OPC UA Information Modeling

```
(rdflib.term.Literal(u'linkedfactory.iwu.fraunhofer.de/linkedfactory/demofactory/machine1/sensor5'),)
(rdflib.term.Literal(u'AnonymousIdentityToken'),)
(rdflib.term.Literal(u'When the action triggering the event occurred.'),)
(rdflib.term.Literal(u'linkedfactory.iwu.fraunhofer.de/linkedfactory/IWU/RolleX/PowerMeter'),)
(rdflib.term.Literal(u'Reports diagnostics about the server.'),)
(rdflib.term.Literal(u'ns=1;s=root_Demo_Scalar_SByte'),)
(rdflib.term.Literal(u'A numeric identifier for an object.'),)
(rdflib.term.Literal(u'i=2403'),)
(rdflib.term.Literal(u'Pure Python Client'),)
(rdflib.term.Literal(u'ns=2;i=1075791275'),)
(rdflib.term.Literal(u'i=11891'),)
(rdflib.term.Literal(u'i=3181'),)
(rdflib.term.Literal(u'i=290'),)
(rdflib.term.Literal(u'i=3094'),)
(rdflib.term.Literal(u'ns=1;s=root_linkedfactory.iwu.fraunhofer.de_linkedfactory_demofactory_machine2_sensor7_value'),)
(rdflib.term.Literal(u'The type for non-looping hierarchical references that are used to define sub types.'),)
(rdflib.term.Literal(u'i=11737'),)
(rdflib.term.Literal(u'An object that represents a file that can be accessed via the server.'),)
(rdflib.term.Literal(u'i=298'),)
```

Listing 0.3: An answer from generated OPC UA Semantic Data

A.10 Transformed RDF/XML Data into Turtle RDF

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix : <http://opcfoundation.org/UA/2011/03/UANodeSet.xsd#> .  
  
<unknown:namespace> :UANodeSet <unknown:namespace#UANodeSet> .  
  
<unknown:namespace#UANodeSet> :NamespaceUris  
<unknown:namespace#UANodeSet/NamespaceUris> .  
  
<unknown:namespace#UANodeSet/NamespaceUris> :Uri  
<unknown:namespace#UANodeSet/NamespaceUris/Uri> .  
  
<unknown:namespace#UANodeSet/NamespaceUris/Uri> rdf:value  
"http://opcfoundation.org/iwu/DynamicServer" .  
  
<unknown:namespace#UANodeSet/NamespaceUris> rdf:_1  
<unknown:namespace#UANodeSet/NamespaceUris/Uri> ;  
:Uri <unknown:namespace#UANodeSet/NamespaceUris/Uri_2> .  
  
<unknown:namespace#UANodeSet/NamespaceUris/Uri_2> rdf:value  
"http://opcfoundation.org/UA/Diagnostics" .  
  
<unknown:namespace#UANodeSet/NamespaceUris> rdf:_2  
<unknown:namespace#UANodeSet/NamespaceUris/Uri_2> .
```

Listing 0.4: Preview of Generated Semantic Data from an OPC UA Server

A.11 Transformed XML Data into RDF/XML

```
<UAVariable BrowseName="0:ServiceLevel" DataType="Byte"
MinimumSamplingInterval="1000.0" NodeId="i=2267" ParentNodeId="i=2253">
    <DisplayName>ServiceLevel</DisplayName>
    <Description>A value indicating the level of service the server can
provide. 255 indicates the best.</Description>
    <References>
        <Reference ReferenceType="HasTypeDefinition">i=68</Reference>
        <Reference IsForward="false"
ReferenceType="HasProperty">i=2253</Reference>
    </References>
    <Value>
        <uax:Byte>255</uax:Byte>
    </Value>
</UAVariable>
<UAVariable BrowseName="0:Auditing" DataType="Boolean"
MinimumSamplingInterval="1000.0" NodeId="i=2994" ParentNodeId="i=2253">
    <DisplayName>Auditing</DisplayName>
    <Description>A flag indicating whether the server is currently
generating audit events.</Description>
    <References>
        <Reference ReferenceType="HasTypeDefinition">i=68</Reference>
        <Reference IsForward="false"
ReferenceType="HasProperty">i=2253</Reference>
    </References>
    <Value>
        <uax:Boolean>false</uax:Boolean>
    </Value>
</UAVariable>
<UAVariable BrowseName="0:EstimatedReturnTime" DataType="DateTime"
MinimumSamplingInterval="1000.0" NodeId="i=12885" ParentNodeId="i=2253">
    <DisplayName>EstimatedReturnTime</DisplayName>
    <Description>Indicates the time at which the Server is expected to be
available in the state RUNNING.</Description>
    <References>
        <Reference ReferenceType="HasTypeDefinition">i=68</Reference>
        <Reference IsForward="false"
ReferenceType="HasProperty">i=2253</Reference>
    </References>
```

Listing 0.5: Transformed XML Data into RDF/XML

A.12 Question Classification

//Buna diger methodlari da ekle

//SVC Gamma Auto

//Rbf Kernel

Parameters	Precision	F1	Recall
Newton-cg	%95.55	%95.56	%95.57
Linear SVC	%92.75	%92.76	%92.77
Limited BFGS	%94.21	%94.22	%94.23
Logistic Re- gression CV	%95.63	%95.63	%95.64
Linear SVC for Li&Roth Taxonomy	%65	%45.5	%35

Table 0.3: The Question Classification of Li&Roth and Wh-Question Taxonomy

Logistic Regression with newton-cg: Logistic Regression is a predictive analysis method that uses a binary classification method wrapped the combination with range $[0, 1]$.

Logistic Regression with lbfgs: Limited Memory BFGS is an optimization algorithm of Newton-methods. We should understand what the Broyden-Fletcher-Goldfarb-Shanno method is

Logistic Regression with Cross-Validation: To classify more than one categories, multinomial logistic regression method. Regression models are useful for continuous data; however, can be used when required a categorical dependent variable. Cross-validation defines the same data set as training and test data.

Linear Support Vector Classification: Support Vector Machine is a type of supervised machine learning which has two advantages over the regression methods. Firstly, SVM creates high generalization performance in high dimensional features. Secondly, SVM can manage a kernel function without affecting the computational complexity in higher

dimensional features. If the kernel function is a linear function, SVM turns out to be Linear Support Vector Machine.

Logistic Cross-Validated Regression gives the better result under a 1559-lined labeled dataset. The Linear SVC has been used to train Li&Roth taxonomy, but unexpectedly the accuracy, precision and F1 Score values gave a lower performance. On the other hand, the Linear SVC created a result over %90 for the dataset of Wh-Question Taxonomy

Appendix B Web Application

B.1 Programming and Architectural Design Pattern in the Web-Based Software

Scalability, modularity, and ease of use are substantive elements of web architecture for a web-based software application in smart factories. Generally, applications that have targeted particular tasks are having a tendency in developing as desktop applications or built-in applications of embedded systems. This type of use reduces mobility for human operators and experts in a smart factory. Hence, a web-based software can be a lifesaver to make the mobility of tasks possible. Whereas, a web-based software cannot be realized without architectural thinking to deploy into a large-scale application. Nevertheless, we should investigate data-intense solutions like the operator assistant web-based software regarding robustness, scalability, high performance and security.

Architectural Pattern: The architectural patterns used for connecting different element of web-based software to introduce a user experience to end users. We will give details about the architectural patterns that we have analyzed.

Monolithic Architectural Pattern: A monolithic architectural pattern can be composed in a single piece of software. In monolithic architecture, we cannot scale components differently and interdependency of components are more potent than the n-tier architectural pattern and microservice architectural pattern. Deployment is the main feature of a monolithic application. A monolithic application can consist of several services or components but if it only can be deployable into a logical structure as a whole, then we can accept this solution has a monolithic architectural pattern. The most significant disadvantage of monolithic applications is that they might not be usable modular so that one cannot solve the complexity of the code base problem easily. However, the main advantage of this architectural pattern is that developers can develop their application in focusing on building a single application. A monolithic architecture can balance internal services by running several instances with a load balancer.

Microservice Architectural Pattern: Microservices are a new design paradigm that has entered to software world by significant features. Deployment may create many difficulties that one has not faced when an n-tier or monolithic architecture. Each service should be small and easy to maintain in microservices. The components of microservices

store their data source such as a database, necessary configuration files distinctly. A significant advantage of a microservice decomposes a whole application into services. This architecture employs lightweight protocol such as REST or Remote Procedure Calls [71].

Although microservices are had advantages over other architectural patterns that have been mentioned in the section, there are several drawbacks that one should consider them. Adoption of a microservice from different architectural patterns, is hard since a system should be separated services and components to minimize in deploying as a microservice. Testing is much harder than others because end-to-end testing and optimal service testing should be considered separately. Distributed applications in microservices may have continuous deployment and broker-based message application to increase deployment and performance features. Such features may cause more complicated architecture when deployment has been initiated.

Microservices decouples the development and deployment process. Such development process could be API development with REST endpoints. For instance, when a developer realizes a web-based software, it would have a dependency from older versions. Older version issue creates a problem for future development. If there is strong communication architecture between components, dependency problem can be enlarged, and the overall system may lose the feature of microservice out. Fault isolation is strong in microservices, which one component does not affect from an external failure as another component has generated.

As a result, the primary principle is dividing the component as much as possible unless it can utilize their data source and internal functions without dependency on other components.

Distributed Monolithic Architectural Pattern: This pattern denotes an intermediary architecture between microservices architectural pattern and monolithic architectural pattern. Components of the distributed monolithic architectural pattern may have a dependency on deployment and development. Binary dependency among components is as strong as monolithic architecture so that microservices can be distinguished from monolithic architectures.

Failure can affect multiple components because there might not be a clear separation between distributed components. Distributed components have not well-separated data

storage and small units that may be scaled independently without having scaled entire architecture.

N-tier Architectural Pattern: Basically, the N-tier architecture comprises presentation tier, application tier, and business tier. Each tier can have multiple layers that can coherently work in a physical system. While a tier represents the physical environment of the architecture, a layer represents more likely abstraction of a tier. Each layer can be a logical organization of the project, which means an organization of code. Layers can connect to each via sockets and sockets are the type of interprocess communication. Layers should be differentiated using multiple sources in tiers. However, there is no need for strict separation for tiers. For instance, operating systems such as Linux and Windows can transfer messages through shared memory. However, the conventional view of architecture represents a single tier.

The Presentation Layer displays information related to incoming requests by end-users in having a relationship with the application and data layer. The Application Layer has a core logic of a system, but the presentation tier is responsible for displaying information to regarding users. It is the last point for all other tiered architecture that introduces the information to end-users. The Data Layer is responsible for containing persistent data in storage. This architecture reduces the deployment cost and easy to optimize in the context of code base. Unfortunately, it suffers from communication points as n-tier architecture increases communication points, which it causes complexity of communication point. In addition, communication parts can share among application layer and business layer so that one can consider that it does not solve separation of layers.

Service Oriented Architecture: “Service-Oriented Architecture (SOA) is an architectural style for building systems based on interactions of loosely coupled, coarse-grained, and autonomous components called services.” [72]. Each service discoverable addressed called endpoints and transmitted composed messages to each other [72]. It supports organization of distributed components in a black-box process. This black-box process utilized heavyweight-XML based protocols such as SOAP or Web Service Definition Language. Despite the fact that it provides a decomposing way for organization with a service provider, a service discovery and a service consumer, it is widely thought as a larger monolithic application. SOA supports loose-coupling and service reusability such as Microservices, but there might be data storage could be common storage whereas microservice discriminates data storage to regarding services.

Programming Design Pattern: This concept stands for repeatable solutions for common problems that one encountered in occurring issues. Even though there are numbered programming design pattern under several categories, we will introduce some design pattern that we have realized in the following statements:

Model-View-Controller Pattern states all parts of software should be dissected separately in having a relationship with the user interface. This pattern decouples the components as model, view, and controller to provide code reusing and separately deployment.

Model encapsulates business logic and data. In computer science, business logic is the part of a program that encodes real-world requirements in terms of creating, reading and updating. All of the items have dynamic nature in an application so that other layer of an application may concern changes that are presented by a model. View demonstrates a view of the modeling of data presented by the same data. It also closely relevant to visualization, beyond that it has a purpose for showing multiple views regarding the same data modeling. The controller acts on both the model and the view. It also copes with data changes and provides an endpoint for a view to visualize data's content.

Client-Server Pattern: Client Server Pattern exemplifies that workload can be distributed between service providers and service requesters. Such servers can save states of sessions that have identified by clients or servers purely communicate with clients in a stateless and cache-based way. Servers hold application logic to present information and calculation when a client requested. A client has responsibility for end users since a client can initiate exchanging messages between servers and clients.

Moreover, a client can send a message instantly to servers to check their health status whether a server is in service or out-of-order. This mechanism called polling that can send a message from client to server within a particular time arrival. Two important concepts differ in that respects. Synchronous communication refers to dependent communication respective messages in a blocking manner. Such messages should wait for the next one to complete their operations. One task can execute an operation as soon as another task finishes. Asynchronous communication concerns repetitive or respective tasks should not be aware of one another. For instance, Task A may execute an operation before Task B does, but Task B can respond to the client before Task A finish the operation. In this way, a user can send a request to a web service without waiting for the

result of queries. The Asynchronous communication is implementing by multi-thread management, finite state machine or event-based callback functions. For instance, event-based callback functions that script language used is a general type of asynchronous function.

Regardless of being asynchronous or synchronous requests, a question might have asked by readers. How can a web-based architecture adapt to increasing workload? The questions leads us to a concept for distribution of workload via a medium. This medium called load balancer.

Single Page and Multi Page Applications: These two programming pattern deeply helpful for developing front-end applications. Single page concept refers to load and refresh all web elements in a web browser during use. It does not refresh entire page but only demanded content. The single page applications can cache local data, which one can load data in case of offline connection. When the offline connection turned out be an online connection, data content updates automatically. The multi page applications update entire page upon each request exchanged between server and web browser. Multi page applications are useful for data driven applications especially like search engine, question answering or database applications. Although it decreases the speed of communication between front-end and back-end frameworks, it increases the scalability regarding requirements of a domain.

B.2 CoffeeScript Sample

```
SDNEntity = require('./SDNEntity.js')
_ = require('underscore')
uuid = require('uuid')
config = require('../config.js')
request = require('request')

class SDNController extends SDNEntity
  controllers = []

  constructor: (@uReg) ->
    @controllerID = 2000
```

Listing 0.6: A sample from Coffeescript

B.3 JavaScript Counterpart of the CoffeeScript Sample

```
(function() {
  var SDNController, SDNEntity, config, request, uuid, _,
    __bind = function(fn, me){ return function(){ return
fn.apply(me, arguments); }; },
    __hasProp = {}.hasOwnProperty,
    __extends = function(child, parent) { for (var key in
parent) { if (__hasProp.call(parent, key)) child[key] =
parent[key]; } function ctor() { this.constructor = child; }
ctor.prototype = parent.prototype; child.prototype = new
ctor(); child.__super__ = parent.prototype; return child; };

  SDNEntity = require('./SDNEntity.js');

  _ = require('underscore');

  uuid = require('uuid');

  config = require('../config.js');

  request = require('request');

  SDNController = (function(_super) {
    var controllers;

    __extends(SDNController, _super);

    controllers = [];
  });
}
```

Listing 0.7: Counterpart of sample CoffeeScript in Figure 1.1

B.4 Script Languages for User Interface Development

Fundamentally, script languages are a subset of programming languages. The front-end framework coherently work with script languages. Notwithstanding, a crucial step sets script languages apart programming languages, which is at the compilation level. Script languages rather interpret where implemented.

JavaScript: JavaScript is an entry point for Rich Internet Application, which provides a content-rich application to an end-user. Event handler concept started with JavaScript language. The functional programming paradigm started with JavaScript for script languages. ECMA Standard is a de facto standard of JavaScript and these properties affect other languages that based on JavaScript language features. Such features include hoisting, callback functions, promises, lazy evaluations, generators, asynchronous iteration, and advanced regular expressions.

TypeScript: TypeScript is an object-oriented version of JavaScript and pretty much closer to JavaScript language. According to the essential characteristics of JavaScript, it should provide a functional language that supports callback functions without object and class. Javascript libraries can be compiled within Typescript language without occurring any problem. TypeScript sets apart from JavaScript with an object-oriented paradigm, static typing, and compile time type checking. Thus, TypeScript is a statically typed language because it should compile all types in compile-time to check the correctness of data. However, JavaScript is a dynamically typed language, which a dynamically typed language should have an interpreter layer, not a compiled one. TypeScript can implement object-oriented paradigms such as interfaces, classes, abstract classes or objects.

CoffeeScript: It is kind of similar context with JavaScript but it gives a concise and compact structure when compared to JavaScript. CoffeeScript reduces coding time thanks to short-cut version of its language property but the drawback of CoffeeScript is that needs a step to compile from CoffeeScript to JavaScript. After conversion to Coffescript, it makes look complicated than Javascript because of pre-processor codes. Basically, Coffeescript reduces code complexity (Counterpart stuff) but it makes a harder syntactic structure for a JavaScript Developer. While Coffeescript is suitable for small module development which can easily be integrated into a bigger module, JavaScript and TypeScript can handle with a large scale of a code base. It makes available to enable an object-oriented script development like TypeScript contrary to JavaScript language.

PureScript: It is one of the largest cross-compiler support as compared to the last three. PureScript aims to be a language in front-end technologies. Large scalability support is one of the fundamental features of PureScript. It can be used in multiple operating systems, C and Field Programmable Gate Array. As compared to other script languages, PureScript has the largest scalability support for multiple platforms. This script language is based on Haskell Functional Programming Language. PureScript support polyglot programming, for instance, a core part of an application could be written in PureScript as well as JavaScript could be used for another module. By using Records can be handled

with more complex Object structures in PureScript. PureScript can modify Classes and instances with keywords with “class” and “instance” by creating advanced typed-data. The type definition is more enhanced than JavaScript because PureScript can preserve data with keywords and it is an *indentation-sensitive*, unlike other script languages. It provides worker threads, which is reducing the number of threads by putting all into a pool mechanism in order to use in case of need.

B.5 Dynamic Server quasi OPC-UA Server

This service defines the way of communication by using message-oriented architecture or standard transmission protocol of Open Systems Interconnection. By utilizing a middleware, publisher and subscriber can be de-coupled. Subscriber or Publisher may be an OPC UA Server to transmit information to clients; however, a severe weakness with this method is a necessity to write values temporarily to the address space. Due to the installation of broker infrastructure, message-oriented service brings more cost to any architecture. In the literature, middleware of Publish/Subscribe Service breaks up two various titles, which is Broker-Based Middleware and Broker-less Middleware [73]. It is generally accepted that the broker-based middleware has a common use in industrial internet of things. The fundamental characteristics of the broker-based are detaching different protocols from each other through a broker, confidentiality between publisher and subscriber, and integrity can be ensured among publisher-subscriber pairs.

Dynamic Server uses message-oriented architecture, and it behaves as a publisher. Hence, it designed differently from other OPC UA Server to takes messages that can be dispatched to particular receivers. Although the fundamental data collector is eniLINK with sensors, Dynamic Server sends a list of values with timestamps and topics to subscribers conveniently. This communication occurs asynchronously because synchronous communication is not suitable for a broker unless it has a non-blocking input-output queue. Through the message broker architecture as depicted in Figure 3-3, each flow of OPC UA can transmit via a non-blocking queue, which is the fundamental step for asynchronous communication. As shown in Table 3-1, the pros and cons have been listed so that broker-based architecture reduces the latency of streamed data to store data into any source such as cloud service or real-time database without loss. The generated data set from Dynamic Server defines a structure through its Information Model and this

model contains data from sensors and actuators that connect to eniLINK. Message Broker Service collects data from production and manufacturing system in the smart factory of Fraunhofer in helping the creation of linked data.

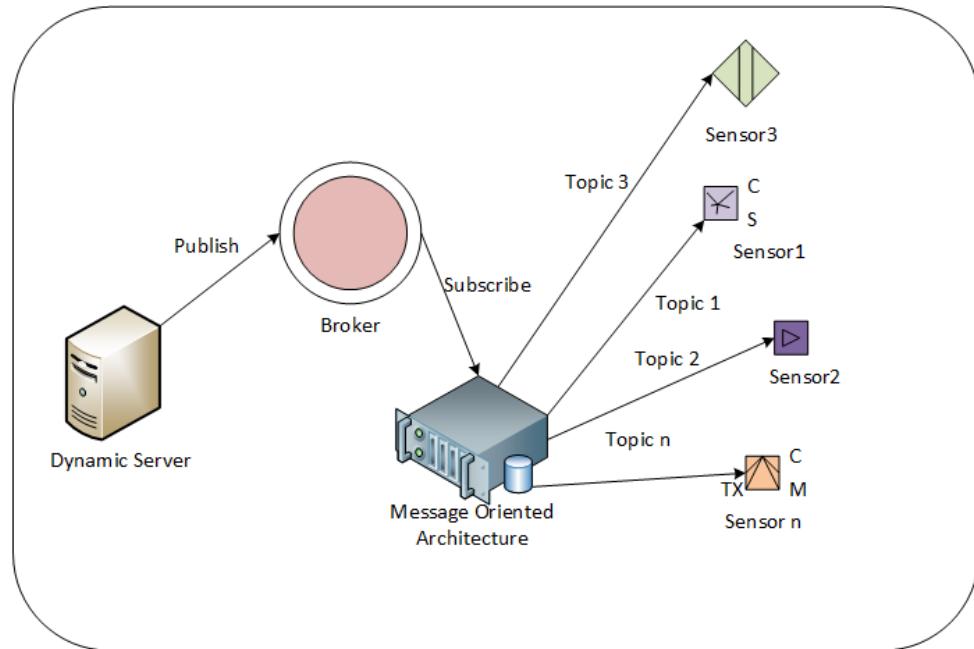


Figure 0.5: The Dynamic Server Publish/Subscribe Model

Types of Middleware	Advantages	Disadvantages
Broker-less Middleware	<ul style="list-style-type: none"> • No Central Point of Failure, • Legacy Devices Support • No additional software components like Broker 	<ul style="list-style-type: none"> • Protocol Dependency because of non-existence a Broker
Broker-based Middleware	<ul style="list-style-type: none"> • Broker reduces latency and overhead generally 	<ul style="list-style-type: none"> • A network bottleneck could be disastrous

Table 0.4: Types of Middleware Publish/Subscribe

B.6 OPC Unified Architecture Information Model Serialization Algorithm

Algorithm 1 Node Extraction

```
1: function MAINFUNCTION()                                ▷ Starting point
2:   export = ServerExport(serverurl, filename)
3:   export.IMPORT NODES(serverurl)
4:   export.EXPORT FILE(outputFile, namespaces)
5: function BUILD NODE TREE(nodes)                      ▷ Node Formatting
6:   client ← GETENDPOINT()
7:   client ← CLIENT(serverurl)
8:   nodecumulated ←None
9:   nodeID ←0
10:  for node < nodes do
11:    nodecumulated = node.nodeid.Namespaceindex
12:    for ref < node.getreferences() do
13:      nodecumulated.extend( ref.nodeid.Namespaceindex)
14:    nodecumulated = list(set(nodecumulated))  ▷ Clear duplicates
15:  return nodeID                                     ▷ Return node id list
16: function IMPORT NODES(serverurl)                   ▷ Traverse Node
17:   client = Client(serverurl)
18:   client.connect()
19:   for ns < client.getNamespaces() do
20:     namespaces[client.getNamespaceIndex(ns)] = ns
21:   root = client.getRootNode()
22:   child = client.iterateChildNodes()
23: function EXPORT FILE(outputFile, namespaces = None) ▷ Export into
   XML
24:   if namespaces != None then
25:     for node != nodes do
26:       if node.nodeid.namespaceindex is namespaces
27:         nodes = [node]
28:       else
29:         nodes = list(nodes)
30:
31:   export = XmlExport(client) then
32:     export.BUILD NODE(nodes)
33:     export.appendXML(outputFile)
34:
```

Figure 0.6: Extraction Algorithm of OPC UA Address Space. The source code can be found: <https://github.com/zointblackbriar/QuestionAnswering>

B.7 Backend Framework Comparison

Criteria	ASP.NET Core	Spring IO	Django Framework	Node.js	Flask
Multiple Queries	46.4%	13.6%	%3.63	24.9%	21.7%
Latency of Multiple Queries	0.5 ms	80.9 ms	287.8 ms	44.4 ms	226 ms
Platform Support	All Platform	All Platform	All Platform	All Platform	All Platform
JSON Serialization	80.8%	%8.7	%10.8	%46.7	12.2%
Latency of JSON Serialization	0.6 ms	4.8 ms	5.8 ms	0.9 ms	3.8 ms
Single Queries	54.9%	12.8%	3.7%	28.7%	10.8%
Latency of Single Queries	0.5 ms	3.8 ms	1.4 ms	0.4 ms	3.5 ms
Plaintext Query	99.7%	2.3 %	2.1%	12.7%	4.1%
The latency of Plaintext Query	1.4 ms	397.7 ms	24.1 ms	65.9 ms	45.8 ms

Compatibility	Backend Compatible with minor versions	Backward Compatible with minor versions	No Backward Compatibility between Python 2.7 and Python 3.0	Backend compatible with major versions	No Backend Compatible
----------------------	--	---	---	--	-----------------------

Table 0.5: Backend Development Framework [65]

B.8 Frontend Framework Comparison

Feature	Angular 6	React	Ember.js	MeteorJS	VueJS
Dynamic UI Binding	B2	B3	Doldur	Doldur	Doldur
Reusable Component	Component-based	+	+	Template-Based	Doldur
Routing	Static Routing	Static Routing	Static Routing	Static Routing	Static Routing
Data binding	Two-way binding	State binding	One-way binding	Template Binding	Two-way binding
Feature Advantage	Object-oriented script development, Independent Library Dependency, Token Interceptor	Doldur	Doldur	Doldur	Doldur
Dependent Pattern	Model-View-Component	Component-based Archute	Doldur	Doldur	Doldur

Table 0.6: The front-end frameworks

B.9 Load Balancer & Reverse Proxy Configuration

```
http {
    proxy_connect_timeout 300s;
    proxy_read_timeout 300s;
    server {
        listen 80;
        access_log off;
        sendfile on;
        sendfile_max_chunk 512k;
        return 444;
    }
    server {
        ssl_certificate ../cert.pem;
        ssl_certificate_key ../cert.key;
        location / {
            try_files $uri $uri/ @mongrel;
        }
        location /integratedstaticmessage {
            try_files $uri $uri/ @questionmodule;
        }
        location /integrateddynamicmessage {
            try_files $uri $uri/ @questionmodule;
        }
        location /fraunhoferengine {
            try_files $uri $uri/ @questionmodule;
        }
        location @mongrel {
            proxy_http_version 1.1;
            proxy_set_header Authorization $http_x_api_token;
            proxy_pass http://backend;
        }
        location @questionmodule {
            proxy_http_version 1.1;
            proxy_pass http://questionanswering;
        }
    }
}
```

Listing 0.8: Sample configuration of a load balancer

B.10 HTTP Headers for Evaluation

```
Connection: keep-alive
Referer: http://localhost:8081/opcua
Accept: application/json, text/plain, /*
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmlxdWVfbmFtZSI6IjEiLCJuYm
Yi0jE1NDgzMzk1NDYsImV4cCI6MTU0ODk0NDM0NiwiWF0IjoxNTQ4MzM5NTQ2fQ.sd
GUDmdgHdPdt37cUyTG1DacVLzJkqcTpPDXSWqHbr8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98
Safari/537.36
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
```

Listing 0.9: HTTP Header of Load Test

B.11 HTTP Requests without Load Balancer

- 1) http://localhost:4000/api/serverconf/1/allnodes
- 2) http://localhost:4000/api/serverconf/1/allnodes/3-
DoubleAnalogDataItemWithEU-Definition
- 3) http://localhost:4000/api/serverconf/1/allnodes/0-2258
- 4) http://localhost:4000/api/serverconf/1/allnodes/1-Matrix
- 5) http://localhost:4000/api/serverconf/1/allnodes/0-2994

Listing 0.10: Multiple Requests without Load Balancing

B.12 HTTP Requests with Load Balancer

- ```

1) http://localhost:80/api/serverconf/1/allnodes/3-
 DoubleAnalogDataItemWithEU-Definition
2) http://localhost:80/api/serverconf/1/allnodes/0-2258
3) http://localhost:80/api/serverconf/1/allnodes/1-Matrix
4) http://localhost:80/api/serverconf/1/allnodes/1-Pressure
5) http://localhost:80/api/serverconf/1/allnodes/0-2994

```

Listing 0.11: Multiple Request with Load Balancing

## B.13 Question Answering HTTP Request

- ```
1) http://localhost:5000/integratedstaticmessage
```

Listing 0.12: Question Answering HTTP Request

B.14 Examples of Discovery Service

Application Name	Advantages	Disadvantages
OPC UA C++ Unified Architecture	Very fast while registering service and finding endpoints Legacy Device Support Automatically initialized by Windows Service Manager	No platform independence Mandatory to install an external communication stack
Node OPC UA Discovery Service	Easy to use with web technologies Communication stack embedded into the node.js modules	Lack of automatically endpoint discover Realized as a desktop Application

B.15 Screenshot of the Subscription Request

```
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:16 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:17 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:18 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:19 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:20 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:21 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:22 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:23 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:24 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:25 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:26 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:27 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
DataChanges: Opc.Ua.NotificationMessage, {17-Mar-19 8:01:52 AM | 17-Mar-19 8:02:28 AM | Running | Opc.Ua.BuildInfo | 0 | }, Good, Opc.Ua.Client.MonitoredItemStatus, 13, i=2256
```

Figure 0.7: Subscription Request with a Monitoring Node

Appendix C Literature Review

C.1 Question Answering Summary

Study	Method	Content & Result
[Molla, Gonzalez Et al. 2007]	Literature Review on Restrict Domain for QA	Potential haphazard of restricted domain QA. Specifications of Restricted Domain QA
[Tirpude, Alvi 2015]	Answer processing, document processing, and answer selection modules for law documents Scoring answers with a module	Statistical results as below F1-Score = 0.62, Precision = 0.92, and Recall = 0.62
[Nguyen, Kosseim 2004]	TREC Dataset, Term Score System for particular keywords	Criticize the WordNet Similarity, Results with Okapi Formulation 53.8%
[Dwivedi, Singh 2013]	TREC, CLECT, NTIRC, All domains in QA, linguistic approach, statistical, pattern-based, template-based	Analyze with significant properties
[Tatu Et al. 2016]	Plain-text and biomedical ontology domain, methods POS tagging, parsing, lemmatization, answer ranking	232, 585 n-triples with mean reciprocal formula

Table 0.7: Question Answering Summary

Glossary

Softmax Layer: It is a regression-based result to assign a multi-classification machine learning problem.

Inter-process Communication:

Machine Learning: It is the science of getting computers to act without being explicitly programmed.

Reinforcement Learning: It is a type of Machine Learning Algorithms which allows software agents and machines to automatically determine the ideal behavior within a specific context, to maximize its performance.

Long Short Term Memory: LSTM is a unit of recurrent neural network which composed of a cell, an input gate, an output gate and a forget gate.

Bi-directional Long Short Term Memory: A bidirectional LSTM layer learns bidirectional long-term dependencies between time steps of time series or sequence data.

Word Vector Representation: It is a word vector in a row of real valued numbers

Recurrent Neural Network: It is a subclass of artificial neural network where connections between nodes from a directed graph or directed acyclic graph along a sequence.

Stanford CoreNLP Tokenization: It provides a tool that tokenizes a text snippet or blob of text

Stanford CoreNLP Part of Speech Tagger (POS Tagger): It provides a tool of which labels tokens with their part of speech tag

Neural Machine Translation: It is an end-to-end learning approach for automated translation, with the potential to overcome many of the weaknesses of conventional phrase-based translation systems.

Epoch: This term explains that is single pass through whole training dataset

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Chemnitz, den 18. March 2019

[Comments] Orcun Oruc

TODO: Es wird empfohlen die offizielle Selbständigkeitserklärung des ZPAs zu verwenden: <http://www.tu-chemnitz.de/verwaltung/studentenamt/zpa/formulare/Allgemein/allgemein/selbststaendigkeitserklaerung.pdf>