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 Question Answering in the Domain of Smart Factory

Orcun Oruc

Chemnitz, 29 May 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, 29 May 2019

Sperrvermerk

Diese Master Thesis enthält vertrauliche Daten des Fraunhofer IWU. Eine Veröffentlichung dieser Arbeit, auch auszugsweise, ist ohne ausdrückliche Genehmigung des 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 have enabled data collection and this concept incarnated as a smart factory in industrial plants.

Nowadays, a factory generates more obscured data from the 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 standard 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 to human operators. Even though a smart factory generates a massive amount

of data through industrial processes, technical personnel or operators cannot easily interpret the linked data created by different sources. A semantic question answering can reply to questions of the operators 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 ease of use. In essence, the goal of this thesis is to create an operator assistant web-based software. In order 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. This research will be an innovative tool with significant findings for future investigations in the sense of human-machine application integrated into the OPC Unified Architecture.

Acknowledgments

The Fraunhofer IWU supported this research. I would like to express the most profound appreciation to my university supervisor Dr. Frank Seifert who provided insight and expertise that greatly assisted the research. I especially thank my supervisor and colleague M.Sc. Adrian Singer. Without his guidance and help, this thesis would not have been possible. I thank Dipl.- Inf. Ken Wenzel for assistance with LinkedFactory and eniLINK Systems [1] and for his comments that remarkably improved the manuscript. For reviewing the thesis and insightful pieces of advice, I owe a debt of gratitude to Dr. Anke Stoll as well. This journey would not have possible without my family, and I am deeply grateful to my family who supported whatever it costs through all my life.

Table of Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

NLP	Natural Language Processing
HMI	Human Machine Interaction
REST	Representational State Transfer
JSON	JavaScript Object Notation
JWT	JSON Web Token
OPC UA	Open Platform Communication Unified Architecture
Fraunhofer IWU	Fraunhofer Institute for Machine Tools and Forming Technology
RDF	Resource Description Framework
SOA	Service Oriented Architecture
OLE	Object Linking and Embedding
SDK	Software Development Kit
SPARQL	SPARQL Protocol and RDF Query Language
W3C	World Wide Web Consortium
XML	Extensible Mark-up Language
DOM	Document Object Model
MVC	Model-View-Controller Pattern
SDK	Software Development Kit
HTTP	Hypertext Transfer Protocol

API	Application Programming Interface
QA	Question Answering
LDS	Local Discovery Server
GDS	Global Discovery Server
URI	Uniform Resource Identifier
IRI	Internationalized Resource Identifier
XSL Transformation	Extensible Stylesheet Language Transformation

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 much generated data and complex protocol structures that can be intertwined dependencies between vendors of devices and facilities in an industrial plant. Advanced manufacturing evolved with technology is 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 the 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 the researcher ignores 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.

The researcher wants to design and implements web-based software to solve the problems above. This web-based software 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 wıth the architectural design to diminish the problems mentioned formerly 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, the researcher should be able to traverse through 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, the researcher aims to design and implementation of an operator assistant web-based software that can answer the limited types of questions related to pieces of equipment of manufacturing and traverse data among various industrial devices through OPC Unified Architecture with limited knowledge of the underlying structure.

Lastly, Fraunhofer IWU continuously conducts research on connected sensors, and actuators' data, which brings scientists the 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 assistant web-based software orchestrates an industrial automation protocol and the natural language processing, which can increase the ability of adaptive automation between human operators and manufacturing devices.

1.2 Problem Statement

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

The problem that the researcher faced is a necessity of an aggregated information extraction-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 the researcher encountered can influence human operators or factory workers who 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 is a significant problem for experts who work in smart factories. Human-centered assistant 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 looking at interlinked data in the heterogeneous data source that consist of streaming data or static data.

The researcher covers 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, the researcher needs to implement the operator assistant web-based software.

Design thinking process of the web-based software can change the robustness and efficiency of the web-based software. Not only the researcher can think of a web-based software deploying for a single manufacturing device of a smart factory, but also the researcher considers that a large number of users can connect to this system. Therefore, robust and scalable design has an essential role in deploying the overall system. In case the researcher obtains a solution, it will be an innovative system that can be implemented into a smart factory, which supports the increase of the efficiency outcome at the manufacturing scale.

The researcher will use a top-down approach to design and implement the web-based software 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 are 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 together subdisciplinary concepts of the computer science the 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, it needs to be clarified 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 is limited to the smart factory 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. This thesis will not focus on the following aspects, which are:

The first limitation is that the researcher will implement a module that allows querying, writing and monitoring for current data assessing the optimal web software architecture at a basic level. The researcher does not aim 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 the 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 sources have been created for the sake of reaching research goals. As of being, a persistent semantic data storage, the so-called 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 that deployment technologies such as containerization or continuous integration technologies will be out of scope. The scope also does not include cloud-based and server-less 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 modifies every possible data types in OPC Unified Architecture; rather, our scope includes designing of the operator assistant web-based software 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 domains with endless types of questions.

Last but not least, the researcher has defined research questions to consider possible implications, and the researcher will try and answer at the end of the research. To reach the primary goal, the researcher has set up research questions and hypotheses. These research questions and hypotheses as below:

The fundamental research question is:

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

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

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 software 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 the scope?

RQ-4 Can the researcher scale the proposed approach to other smart factories or plants?

My hypotheses are the following:

H-1 The modules of the 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 from previous studies 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:

- Introduction of an innovative web-based software by demonstrating a detailed architecture for realizing an assistant software that is robust, secure, performant, and ease of use. The web-based software integrates architectural thinking, the OPC Unified Architecture industrial communication, and natural language processing.
- Proposal of a synthesized idea through transdisciplinary areas of computer science the so-called web science, communication network and artificial intelligence to develop an assistant software for human operators in a smart factory.
- Suggestion of a human-machine interaction tool that can answer to factoid questions from various linked data sources such as time-series data and semantic OPC UA data.
- Proposal of a heuristic-based syntactic analysis to solve the question answering challenges in a restricted domain question answering.
- Introduction of an industrial communication tool integrated into a semantic question answering that is aware of context information in a smart factory.

- Evaluation of the architecture with qualitative and quantitative researches through performance and functional experiments and provides empirical results to the readers. Also, the researcher evaluates the question answering part with manually generated questions.

This study is organized as follows;

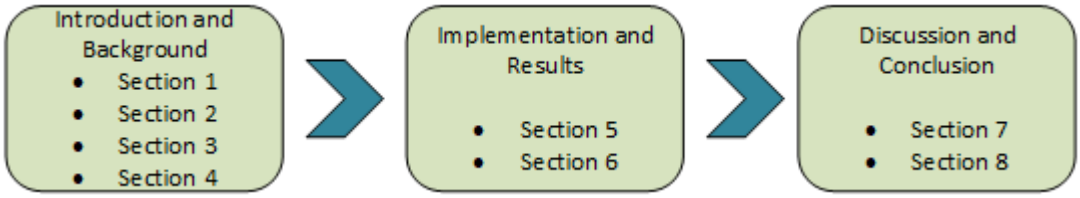

Figure 1.1: Organization of the Thesis

Section 1 presents the motivation, problem definition, scope, and related works.

Section 2 conducts a literature review regarding OPC UA, Question Answering and Linked Data Collection that is examined separately. For OPC UA, protocol level benchmarking and integration into web technologies are examined with advantages, disadvantages and current trends towards human-machine interactions.

Section 3 introduces industrial communication in smart factories in terms of the theoretical background of operator assistant and human-machine interaction. After expositing theory of OPC UA Service Sets, the researcher is creating an insight regarding OPC UA communication.

Section 4 explains the theory of the Semantic Question Answering in the context with natural language understanding for the operator assistant solution. The researcher explains the theoretical background of the basic methods of Natural Language Processing, semantic technology for the semantic QA and data preparation for the heterogeneous data source.

In Section 5, the researcher will elaborate on the details of practical implementation with architectural details. This practical implementation comprises research methods to evaluation for the following sections.

Test methods, datasets, and results will be in Section 6, and the researcher will list the results under the section. Before concluding, the researcher will comment on experimental results and express all findings relevant to the research in Section 7.

Finally, the conclusion part will define the meaning of findings and future improvements in Section 8.

2 State of the Art

The state-of-the-art studies will be discussed in this section by splitting up into 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, the researcher will summarize the current 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. The term named *"Object Linking and Embedding for Process Control"* traced back to its 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]. On the path of development, OLE OPC, which was formerly known as OLE for Process Control has been widely dispersed at the industrial scale.

The principle of OLE is to develop modular applications that refer to a groundbreaking 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 is that they 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, the OPC Unified Architecture has solved the problem of multiple releases by means of implementing 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 from how to enhance security in industrial networks when OPC Classic

was used. Besides, the first data access modeling has been developed only for reading, browsing, 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, which is a platform-vendor independent and service-oriented architecture. The primary motivation for migrating from OPC Classic was that its message protocol based on the Component Object Model, so the 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-orientation, OPC Unified Architecture 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 & Scroppo 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 [7]. 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 they do not implement a front-end application to scale in a web architecture in the context of user interaction and performance. Besides, 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 architectural decision as middleware, a user request module, and an OPC client module. The studied algorithm is reproducible but not easy to configure for testing [8]; nevertheless, it can give readers insight and design decision. The implementation and research have conceptual differences about HTTP method definitions, *"Publish/Subscribe"* implementation and serialization of data. However, [Cavaliaeri, Salafia & Scroppo 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 was 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, which is 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 that covers the *"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 study is that it 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 the REST and the 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 layer [11]. In any case, these two types of research did not aim for the implementation of 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 the Representational State Transfer System by listing the features that they need. The authors emphasized the main problems, which are *"HTTP Mapping"*, *"Session-less Invoke"* and *"Browser Support"* [12]. It is generally accepted that the problem set of [Shiekofer, Scholz & Weyrich 2018] overlapped the problem definitions of the researcher, 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 as to information extraction, information retrieval, and natural language processing theories. The *"Open-Domain Question Answering"* identifies the question that 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. The *"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 request 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, the researcher 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 the researcher initially will focus on researches about the characteristics and features of a semantic question answering in this chapter. Then the researcher ought to examine algorithms and applications that have specified in domain-specific question answering.

Regarding the Semantic Question Answering, researches mostly focus on algorithms on how to transform a natural language expression to the *"SPARQL Query"* and the *"Resource Description Frameworks"*. A semantic question answering can use different types of dataset ranging from structured data to streaming data. Primary data sources are plain-text documents, open data cloud (Wikipedia, DBpedia), time series data, and linked data which consists of semantic triples.

Few researchers have addressed the problem of restricted-domain question answering. [Molla, Gonzalez Et al. 2007] overviewed the main characteristic of a question answering in restricted domains, which is the integration of domain-specific information either developed for question answering or disclosed for other purposes [13]. The authors defined 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, which he 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 a question processing module, a document processing module, and an 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. Test questions have 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 the researcher can take into consideration the precision and the 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 on the 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>"*, the dependencies between the different phrases [17]. The methods as mentioned earlier mostly founded the stage of question analysis. According to this survey, constituency parser on phrase structure, dependency grammar on the relationship among the parsers might be used. N-gram language modeling 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 syntactic or semantic similarity. Lastly, in their conclusion, they have proclaimed that an examination of every existing technique 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 reached with 60 questions to 53.8% accuracy rate under a particular document set [18].

It was introduced characteristics and features of a question answering so far. Later, the researcher will examine the algorithms regardless of being open-domain, closed domain or restricted-domain.

[Dwivedi, Singh 2013] briefly survey the significant characteristics and algorithm types of QA Systems. They have determined a couple of 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, TREC[1], 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, lemmatization, 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

[1] https://trec.nist.gov/data.html

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.

[Giannone, Bellomaria & Basili 2013] examines the Hidden Markov Model (HMM) in conjunction with unsupervised statistical learning. They aimed at mapping and linguistic expressions such as proper nouns onto ontological elements such as DBpedia[2] corpus by implementing lexical similarity [22]. They have used a syntactic chunk-based dependency graph that calculates lexical similarity values between word pairs. Then, the Hidden Markov Model, which based on Markov Chains that computes the probability for sequential observables event from hidden states. They aimed to find relationships between linguistic elements by correlating hidden and observable states. In their evaluation, the HMM method shows lower performance with QALD-3 dataset than [Ferre 2012] *"SQUALL controlled the English language"* in accordance with the F1-Score, the Precision, and the Recall which are 0.32, 0.34, 0.33 respectively [22].

[Unger 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 [23]. 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 [23]. 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 [23]. 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.

[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 [24]. Their architecture consists of tree tagger, WordNet similarity 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

[2] https://wiki.dbpedia.org/dbpedia-version-2016-04

noun/verb/adverb and adjective were checked with WordNet to map onto ontologies [24].

[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 [25]. The *"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 [25]. It has grouped all substantial features of a morphological language and pointed out what type of linguistic features 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 [25]. [Giannone, Bellomaria & Basili 2013] proclaimed the performance of SQUALL against *"QALD-3"* dataset shows the recall, the precision, and the F1-score which are 0.85, 0.89, and 0.87 respectively [22].

[Luz, Finger 2018] aimed to transform natural language expressions into a query of SPARQL. They have used the 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-vectors[3] 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].

The *"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 [26]. The authors proclaimed that a task orientation becomes essential in evaluating a restricted domain question answering. Their data source is 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

[3] https://nlp.stanford.edu/projects/glove/

updatedness of their data source, the user interaction testing specified the testing phase as querying style (Keywords, sentence-based), question formulation assistance that composes a spell checker, an abbreviation recognition and a feedback collection [26].

2.3 Linked Data Collection for Heterogeneous Data Source

Linked Data Collection is not part of one of the research objectives, but the researcher will overview approaches for the sake of the semantic question answering. The researcher 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, the linked data serialization from OPC UA Servers converts the Information Model of OPC UA 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. The information model provides standardization of information representation to be understandable by the several systems. The researcher 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 authors emphasized that the absence of a standardized information model for machines, each vendor may implement their data and information model to devices [27]. Their aim is to create a semantically augmented OPC UA framework that enhances the knowledge in production for decision-making systems [27]. 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 models from different machine-to-machine

protocols [28]. The difference of their approach is creating a uniform information model that can adapt the information in SQL Database, memory-mapped values, *"OWL"* ontology or *"AutomationML"* formats [28]. The difficulty of their approach, that is, a platform should discriminate the containment relations, inheritance relations and type-instance relations of the Information Model to assign to particular ontology items [28]. 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, the researcher will search over the linked stream data processing to create format suitable to linked data. The *"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 [29]. 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 [29]. *"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 providing transformability to multiple RDF sources such as N3, Turtle and automatic assignment of a namespace to be specified on the sensor and actuator applications [30]. 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 [31] :

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 data queries, choosing related datasets in Linked Data Cloud Storage and creating data linkages [31]. 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 values. 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] offer to use Semantic Sensor Networks that can describe capabilities, measurements, and resultant from sensors and actuators [31].

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 [31]. [Llanes Et al. 2016] demonstrates two approaches which are: *"R2RML"* [Calbimont, Corcho & Aberer 2011][32] and *"SASML"* [Zhang Et al. 2015] [31]. 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 the 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 [31].

[Anicic, Fodor 2011] proposed Event Processing SPARQL (EP-SPARQL) as a new language for complex stream events [33]. The primary goal of their proposal is to provide a fundamental framework for Event Processing and Stream Processing [33]. The authors created a new quasi SPARQL language that has some similar functionality such as *"Seq"*, *"Equals"*, *"OptionalSeq"*, and equal options that are used to combine graph patterns in

the same way as *"Union"* and *"Optional"* statements in SPARQL [33]. 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 [33]. C-SPARQL is an older version of EP-SPARQL that consists of RDF streams, *"Windowing"*, *"Registration"*, and *"Aggregation"* [34]. 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"* statements 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 the *"Linked Data"* [35]. 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 the *"Hypertext Transfer 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 [35].

2.4 Chapter Discussion

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

First of all, the OPC-Web Integration has not been overviewed about the 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][36]. The research in [Cavalieri, Salafia & Scroppo] [6] [7] has detailed the *"Publish/Subscribe"* implementation of OPC UA. They created a monolithic architecture with-

out 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 the OPC UA Information Model can be 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; however, 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 essential guide for existed algorithms. [Tatu Et al. 2016], [Celikyilmaz 2006], [Palaniappan, Sridevi & Subburaj 2018], and [Unger Et al. 2012] have employed statistical methods to reserve morphological structure of a language. Advantages and disadvantages of their approach might not be readily observed because every approach is peculiar to its domain and test set. [Giannone, Bellomaria & Basili 2013] offered a framework based on Markov probabilistic approach which is an expanded version of [Celikyilmaz 2006]. The advantage of the study offered by [Giannone, Bellomaria & Basili 2013]

is unsupervised statistical machine learning method realized for question analysis, finding lexical similarities word pairs, and linguistic mapping expressions onto ontology resources as a whole. The disadvantage of the paper is that SPARQL query compilation phase has not been detailed for common cases.

Significantly, [Ferre 2012] has offered controlled natural language expressions 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 of the techniques Knowledge Base Question Answering, which is a quite valuable analysis that can be used in the thesis [17]. This examination consists of the possible use cases concerning with methods. Given headline on the question analysis, phrase mapping, and query construction may serve as useful functions. The advantage of this paper is that the examination has been covered in crucial 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 could 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 the 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 prior studies have not given much detail that the researcher can follow.

3 Industrial Communication in Smart Factories

The fourth revolution of Industry known as *"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 [37]. Regardless of the context of manufacturing, Industry 4.0 has changed communication infrastructure between devices and machinery at industrial communication working groups that have typically aimed in industrial plant requirements, which are increasing efficiency and productivity that have been required. Having a unified digital platform for industrial communication, devices and machinery can communicate with each other no matter what kind of vendor-specific applications. Interactive assistant tools, transformation ability, and efficiency in production are vital factors that can affect the future of the digitalized factory.

A smart factory is a highly digitized and connected production facility that relies on smart manufacturing [38] — this concept one of the critical 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 [39].

The definition of smart factories has been 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. Within the essential aspects, this study informs the readers how the human-machine interaction and smart operators 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 [37]. The cyber-physical system embraces complex networking, integration of embedded systems and application systems, enabled by

human-machine interaction [40]. Smart operators should be equipped to realize the helper system to humans in increasing interactivity and ergonomic design.

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[41]. On the 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 the 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 have been optimized and maintained by human operators. Thus, 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 is 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 the human-machine interaction should provide.

With the improvement in the *"OPC Legacy Standard"*, the industrial revolution achieved the interoperability between heterogeneous devices at the communications level, regardless of the manufacturer [42]. Question answering systems increase the capability of transforming query languages. Semantic structured or relational data was used to show result employing a particular query language SQL or SPARQL.

A web-based solution has an advantage over developing a smartphone application or tablet application following 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 the HMI legacy devices in order that interacts with the data of industrial processes [42].

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, diagnosis on sensors or actuators is not possible without knowing the underlying structure. Nevertheless, a smart operator can help to avoid this type of issues that human operators faced, beyond that the human operator can inform the management personnel about situational analysis. Throughout all process may affect the decision-making process, or at least informative resource would be obtained in a short amount of time.

In this way, 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 a low-level protocol stack and high-level applications on the communication path. Because non-existence of a set of subroutine definitions for the communication stack, services are conforming to their service sets. In order to implement a web-based software, OPC UA Services must be suitable with regarding web architecture. OPC UA Services have original specifications and requirements for each service. Each service has service sets that show abstract descriptions, and they do not implement specifications [43]. These services could be *discovery service set*, *secure channel service set*, *session service set*, *query service set* and *subscription service sets*. The *discovery service* set is used to find an OPC UA Server endpoint that has connection string by conforming automatically or

manually. The *secure channel service set* ensures the confidentiality and integrity by certification exchange among clients and servers [43]. The *session service set* used to create a session with the secure channel service set to reach an address space of a server. The *query service set* handles current and historical data concept to return bulk data from an address space. Lastly, the *subscription service set* has a set of a cyclic poll item called *"MonitoredItems"* defined by clients to notify progressive changes in a particular time interval [43]. So far, the researcher has explained the abstraction layers of OPC UA communication stack for higher-level functions. The following parts will introduce the substantial sections that the researcher examines 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 offers 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. After summoning the OPC UA into action, it 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 in order that clients can browse, read and write using nodes in the address space.

The smallest item in the address space of OPC UA is termed the *"Nodes"* which belong to *"Objects"* [44]. A node comprises *"Attributes"* and *"References"* which can be reached by *"Node Class Browse Name"* [44]. *"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"* [45]. When a user 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 real-time data processing, the address space has a breakthrough feature where the data exchange in a real-time manner. This thesis is a review of a preliminary attempt to explain items of address space which are [46]:

View holds a set of nodes in a living space while sending a request to the address space. **Object** represents real-world objects and software components, and it may use *References* additionally to define relationships of Nodes. **Variable** provides real-time or simulation

values when a client is browsing in it. **Method** items correspond to callable events by returning a state.

The items as mentioned above defined as the general parts of the OPC UA Address Space. There must be data containment when the OPC UA utilizes these items. *"Mandatory"* and *"Optional"* selection is contained in type definition so that one can decide how to apply a type.

Object Type consists of a definition of Objects. **Reference Type** is used for meta-modeling providing an inheritance of objects and defines meanings of a relationship among nodes. **Variable Type** determines some types such as *"Historical Data Access of Variables"*, *"Minimum Sampling Interval"*, *"Access Level"*, *"User Access Level"*, *"Array Dimensions"*, *"Value"*, and *"Object Type"*. The variable type has a vital role in practical implementation because the definitions of Variable Type enables browsing, reading, writing, and subscription.

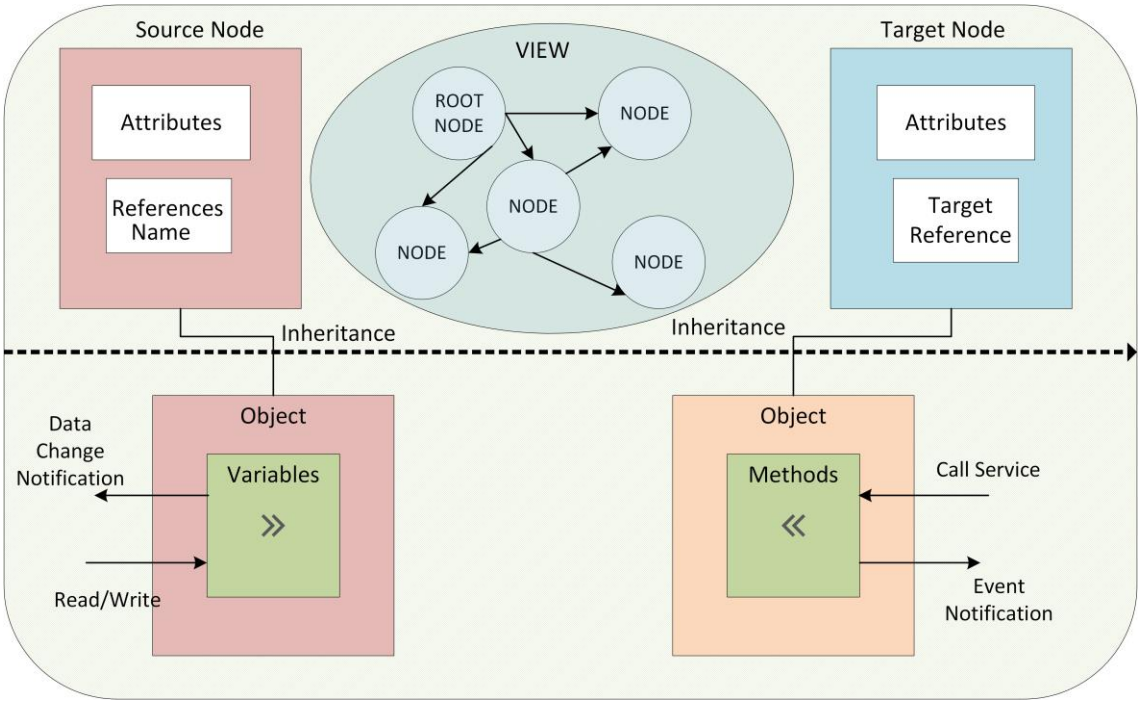

Figure 3.1: OPC UA Address Space [44] [43]

As shown above, in Figure 3.1, 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 one another in an address space

through a restricted address space is called "*View*". In Figure 3.1, the upside region that has separated by a dashed arrow shows an address space and below one demonstrates an information model that inherits from and links into an address space. The *View Service* in OPC UA helps to navigate hierarchical references to search for information about nodes, attributes, and objects of nodes. Lastly, the reading, the browsing, and the writing requests are handling with the *OPC UA Address Space* through the definition of service sets.

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 [46]. Beyond this concept, a semantic modeling tool provides a two-way standardized communication version of the Address Space. Strictly speaking, it is a way of object-oriented representation of servers that can be reached by clients. The main difference between the *"Address Space"* and the *"Information Model"* is suitable with meta-modeling languages, such as *UML* and *SysML*. The "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 the service-oriented architecture so that every object can communicate related service to exchange corresponding data. *"Object Types"* defines types of an object depends 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 *"Fundamental-Scalar Data Type"*, or *"Application-Defined Data Type"*. A web-based software may provide an improvement when a user requires reaching a scalar typed or application-defined typed objects.

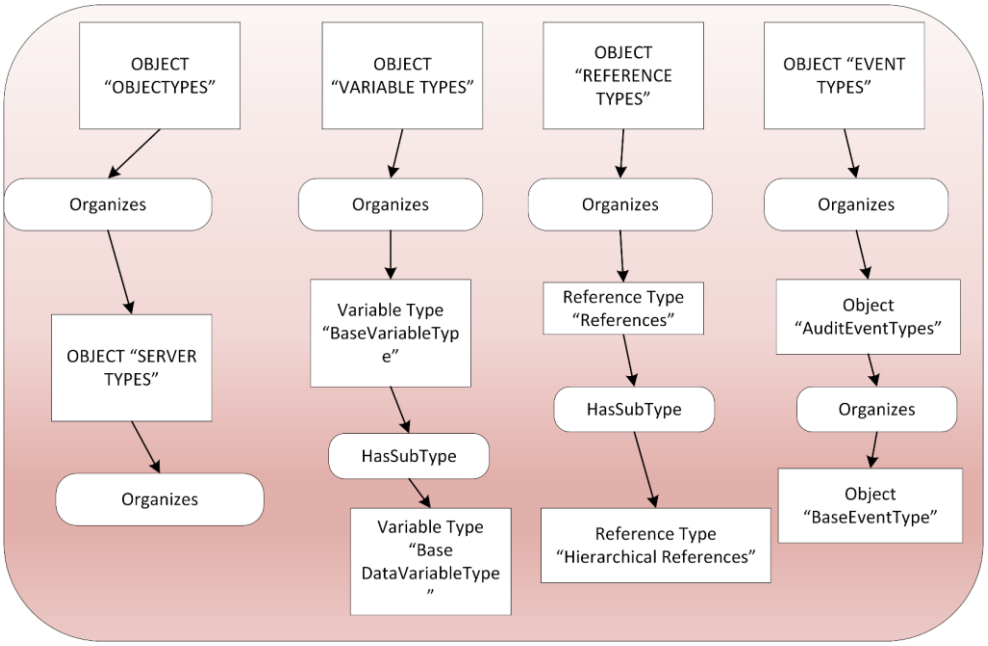

Figure 3.2: OPC UA Information Model [47]

The 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, a 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 identified 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 extensible markup language elements. It is not a mandatory type of definition in references; nevertheless, it is a compulsory schema structure building up a hierarchy in OPC UA. *HasTypeDefinition* is a definitive term for the type definition of the *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* determines 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 [44]. OPC UA Servers have useful items called *"Folders"* that serve as

separator objects that have similar type definitions. *HasProperty* is used to describe *Properties*, which properties have relationships with other features of a reference type. *HasComponent* identifies the data variables, the methods, and objects contained in the *Object Class* [44].

Consequently, each element of the information model defines particular relationships in a hierarchical way or non-hierarchical way. In a general sense, the information model defines types and references, which are the essential component of abstraction. As a result, the web-based software relies on the information model for browsing between nodes with their references 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 service-oriented architecture 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 partly able to use discovery service, and then clear-cut key points are described in service discovery of OPC UA. Overviewing better the discovery process, the researcher 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 locating items of a network by a specified device of a network. For instance, a client can find a server using 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 a connection between OPC UA Clients and Servers. Discovery services 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 [43]. The 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"* [43]. When a client requires establishing a connection, a session is not supposed to be created. Hence, every server has the *"Discovery Endpoint"* to connect clients without creating a session [43]. 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 obtaining a *"GetEndPoints"* service set from a discovery process, it can open a secure channel by providing a certificate, hashed authentication or the anonymous connection to perform opening a communication channel. Accordingly, between finding an endpoint and sending the endpoint requests, there is no authentication schema. Lastly, a discovery service implementation could cause security vulnerability inter-smart factories, but discovery service can make consistent architectures to scalable applications. Advantages and disadvantages of existed discovery service can be found as shown in Table 3.1.

Architectural Decision	Advantages	Disadvantages
Industrial Communication with Discovery Service	Suitable with Microservice and SOA Design Management of certificates with a single point of view	Insecure connection before *"getEndpoints"* between Client and Discovery Server
Industrial Communication without Discovery Service	Suitable with Monolithic Design Less complex architecture No continuous discovery message overhead	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 was implemented on a stateful architecture such as session-based applications, there would be an issue for identifying data alteration. 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 are continuously changing 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 3.3, 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 the OPC UA protocol to simulate data, which are data changes of *Variables*, *Objects of Events* or *Attributes*. The sampling interval is a critical factor of monitorable nodes to detect changes in 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 software send off 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 submit or not. Besides, a filter can eliminate a different type of an item to be monitored so that additional notification cannot overflow in the system. A subscription service put all notifications into a queue that can transfer without blocking any corresponding notifications.

If a new notification has been entered into the queue, a prior notification should be deleted to free the queue size. Monitored items should comply with the minimum sampling intervals. As a result, the minimum sampling interval specifies the degree of the sampling interval, and this minimum value of the sampling differs 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 to fetch correct notifications with a decent value. Taking everything into account, the amount of the smallest minimum sampling interval can create a maximum load of traffics for OPC UA Server, which can occur buffer overflows.

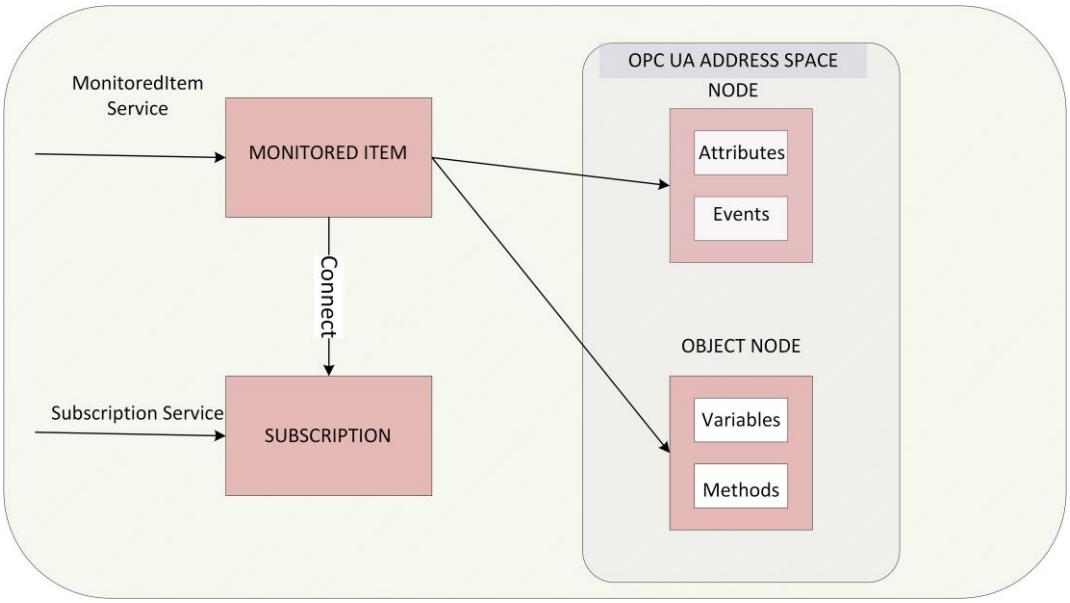

Figure 3.3: Subscription and Monitoring Item Services [43]

3.3 Chapter Discussion

The services of OPC UA 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. In spite of insufficient information modeling of the communication stack in OPC UA, each OPC UA Server realizes its information modeling and address space. Moreover, data that has been stored in OPC UA Servers have various definitions, and it can change the semantic understanding. External connections of the primary functionality of OPC Unified Architecture are organized with *OPC UA Service Sets*. Both the heterogeneous data source of the Semantic QA and OPC UA Web Component get benefits over *OPC UA Service Sets*. *OPC UA Address Space* and *OPC UA Information Model* are in a stable relationship. The web-based software provided three basic features, which are the reading nodes and attributes, the writing variables, and browsing should be compatible with the OPC UA specifications. The discovery service is a complementary module to meet the answer of RQ-1 and RQ-4. The subscription service can complete continuous polling task in the web-based software that related to RQ-2. Finally, yet importantly, a smart operator web-based software discovers useful information by giving predictions according to data in connection with the OPC UA Web Component and the Semantic QA.

4 Theory of the Semantic Question Answering over Linked Data

The question answering is a balancing exchange 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 side, a question answering performs a task on natural queries to detect syntactically and semantically; on the other side; 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 set of collections that can be ontologies or documents. Since the *"Information Retrieval"* and the *"Information Extraction"* have perplexed definitions that one should examine with similar and dissimilar points to give a better insight on question answering systems to readers. The information retrieval is a research term that is used to locate a document that required by a user, but a user defines a relevant answer after obtaining a document. The information extraction is a term for extracting a set of information from a user input so as to learn relationships between searched keywords and documents[4].

According to the functionality of question answering can be categorized in the following way:

Information Retrieval-Based Question Answering has the goal to answer a user's question by segmentation of typed contents of text on collected documents [48]. Main features of the question answering are strongly bounded to the 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 the rudimentary idea that *subject-predicate-object* triples of ontology are able to map onto simple relations of natural language queries. It uses methods such as natural language processing and information extraction intricately.

According to the data source can be categorized in the following way:

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

[4] https://www.ontotext.com/knowledgehub/fundamentals/information-extraction/

plain-text documents for a data source. **Context-Based Question Answering** handles document-based datasets to respond to questions asked by users. This question answering does not interest in ontology data.

According to the type of domains can be categorized in the following way:

Open Domain Question Answering can interpret any topic that a user wants to reach a result from a general domain. This question answering may have a context-based or ontology-based data source. In **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 particular domains such as closed and open domains. **Restricted-Domain Question Answering** is more likely one can ask a question against semantic documents 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 the information retrieval capability is strictly relevant to natural language processing capabilities.

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

According to types of questions can be categorized 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 might be ineffective against a factoid question. Factoid questions may include list questions, wh-typed questions (e.g., what, who, which) or keyword-based search. **Indicative Questions** are 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 essential 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 frequently been a prominent topic for question answering. This work can use a keyword to extract information from the Turtle RDF data format

with a specific keyword. Verbs and their predicate counterparts, nouns and their counterpart objects in RDF are selected as the baseline on Fraunhofer IWU's data source.

Indirect Requests state a query 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 the listing of the desired results. **Boolean Questions** should only have the answers that may be *yes* or *no*. The researcher has 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 a non-factoid question because there are reasoning and induction to find out the 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 series of events. The main reason is that this research did not implement the reasoning induction because the types of questions have complex structures for the statistical model, requiring a vast amount of data, and answer ranking module.

Lastly, the proposed semantic question answering can be considered as 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 that can be used for understanding and interpreting data in representing methods of computer language so that separated domains can solve the problems of external representation. In particular, 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 (RDF) is one of the most ubiquitous data models for interchanging web-based information. As it is understood from its name, it is a framework to support resource description and metadata in the linked data. Each RDF member represents as triples, and each triple might have connections 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 requirements[5]. The RDF consists of several types of models that currently dominates in industry. The fundamental part of an RDF data is a prefix the so-called *Uniform Resource Identifier* so that query languages navigate internal structure with URI. Resource Identifier is not feasible to all kinds of generated documents from the extensible markup language. The primary purpose of the RDF is to be an interpretable machine medium for linked data in the *World Wide Web*. The algorithmic representation of RDF is a graph data structure, which has a set of vertices and edges.

Serialization denotes 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. An example is a blank node namespace defined by an underscore, but regular namespace should not use that underscore. The primitive types of XML Schema must be compatible with RDF types. In the case that the XML Schema types have incompatibility problem, there might be an increase in the number of blank nodes or undefined type values so that it could degrade the feasibility of a question answering.

Besides**, the Turtle** is a reliable alternative for RDF/XML; the syntax of Turtle Semantic is similar to SPARQL queries. The *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 complex 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 supports underscored namespaces. Syntactically, N3 triples are a subset of Turtle RDF because they were designed to be a simple format than Turtle RDF. *Notation 3* demonstrates similar syntactic definitions to Turtle semantic format; nevertheless, a group of differences to Turtle RDF have been observed. Triples follow the patterns of <*subject-predicate-object*> and terminal notations. Notation 3 has enlarged grammar structure with extra features more than *Turtle RDF* and *NTriples*.

[5] https://www.w3.org/standards/techs/rdf#w3c_all

N-Triples Parsers and serializers can easily parse this format because simplicity of syntax definition in triples. There are no complicated grammar rules in the *N-Triples*, but it is not a human-readable format. The *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>"* that contains white spaces and dot-separated values. It is not a compact semantic format which has not abbreviation feature that makes it hard to read by humans. In addition, N-Triples must do extra labeling operation for blank nodes.

JSON-LD provides a lightweight linked data format, so objects should then be converted to a human-readable form. This format is a compact format that has compliance with the JSON data. The JSON-LD format has a strict dependency on the JSON format, and it can be used without considering prior information about the RDF. Typically, the JSON-LD contains the same structure as compared to RDF like primitive types for nodes and *Internationalized Resource Identifier* (IRI) definition of edges. The significant advantage is that standard parsing methods for JSON can be used for the JSON-LD interchangeably.

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 is used in a linked data. This ontology language leverages RDF Schema to recognize complex knowledge requires complex properties [49]. Despite the fact that OWL has a complex structure, the OWL can make classes the properties that co-occurrence with RDF labels.

SPARQL Query Language (SPARQL) is a specialized query language that utilizes a particular protocol endpoint for performing data manipulation from RDF datasets. RDF has a collection of graphs, and these graphs are directed and labeled. Thus, triples of graphs can be obtained with a query language from databases or files. The structure of SPARQL resembles somewhat Structured Query Language (SQL), but the SPARQL was designed for the use of semantically structured triples, not for relational datasets. Additionally, the 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 are used to find the

location of objects. IRIs has a broader range of characters to be used in order to accommodate a wider range of languages than URIs [50].

Mainly, *"Remote Queries"* or *"Native Queries"* characterize SPARQL Requests. Remote Queries are determined as if they were sending a query against remote SPARQL endpoints. Remote Queries need an endpoint definition provided by Linked Data Source. As regards the *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 complex queries over to the SPARQL Service. The 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 by providing federate property into queries. The working principle of *"UNION"* is similar to *"Outer Join"* in the SQL. It takes all *"Cartesian Product Multiplication"*, so one can state that the result of an answer impacts the redundancy of triples. To decrease this type of redundancy, the *"UNION"* can be used with an *"OPTIONAL"* statement or query can be optimized with only an *"OPTIONAL"* statement. *"OPTIONAL"* is used for allocating a particular portion of SPARQL into results of triples. *"OPTIONAL"* reduces the redundancy of data and gives every match in any triples. It is common to use *"OPTIONAL"* query with *"FILTER"* that allows measuring up a couple of criteria.

One of the most significant problems occurs between blank nodes while searching triples where the triples are specified without explicit IRIs declarations. A generated Turtle RDF may define blank nodes that have no clear identification. An RDF serializer can do a preliminary process by 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 with each other.

As shown in Appendix A.7, matching meaningful predicate onto names is a crucial step employing with SPARQL queries. Converting from natural language expression into triples, verbs often are mapped onto predicates. Predicates are 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 indicative definition while 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. In this study, the web-based software employs SPARQL queries with Turtle Data Source. For instance, a SPARQL request in Appendix A.8 has been used to fetch properties from generated data.

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 data incoming from the *eniLINK* and regarding hierarchical data. The latter is objects-references mapped static data incoming from a particular OPC UA Server called the Dynamic Server (see Appendix B.5).

This work utilizes a SPARQL query interface that is 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 is a request which is wrapped up by SPARQL validator. This service is an internal development with the *InfluxDB* time-series database and *LevelDB* key-value mapping database. Continuous SPARQL Service was reviewed for streaming data and analyzed architectural differences in Section 2.3.2.

Using a continuous SPARQL language instead, the *"KVIN"* can map semantic data with properties. A predefined namespace was added in the *"KVIN"* Service to convert SPARQL triples compactly. Relationship with other components of *"KVIN"* can be observed as seen in Figure 0.3. The *"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 databases, 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 length of the crossed path in graphs increases the execution time of queries in defiance of the size of graphs in the storage. This is one of the biggest advantages of a key-value database over relational or hierarchical databases.

[51] [52] [46] [53] referenced works have contributed to the implementation of the serialization process from an OPC UA server. However, the application and the algorithm as shown in Appendix B.6 have some conceptual differences to understand better the context. The Serialization of the OPC UA information model has several stages. At the initial step, 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 the root node until reaching terminal nodes. *Build Node Tree* takes nodes with namespace index and *Node ID* by clearing duplicated elements.

The *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 Turtle linked data. The *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 [53]. 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 [53].

4.3 Natural Language Processing in Question Answering

The *Natural Language Processing* is two-folded, which are *Natural Language Understanding* and *Natural Language Generation*. Natural Language Generation is a concept to create from a language description to a formal language description that may constitute a set of formal rules, rules of syntax and semantics. For instance, a neural machine translation system provides a language exchange interface to perform a set of linguistic rules on words in sentences and transforms them into another language. The scope of this work does not address the natural language generation; consequently, this work examines methods of natural language understanding. Nevertheless, the natural language processing will be used as a term in the remainder of this study.

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 languages. 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 with examining a corpus. A corpus stands for the body of texts or collections of documents. Multiple sources of collections are named corpora [54]. Generally, restricted-domain question answering works with collections of texts 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 in terms of question classification.

The following listed methods likely meet the requirements of the semantic question answering.

Normalization is used to eliminate the 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 implementing natural language processing methods. For example, stop word removal is a normalization process. The 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, the NLTK library has an extensive list of stop-words for the English Language, so 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 words or phrases. Tokenization is a practical part of language modeling, and language modeling plays a considerable 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 such as *"lf:contain"*, given verb are cleansed 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 the *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 the lemmatization needs a canonical form of words in dictionaries. The distinction between lemmatization and stemming is that the lemmatization should be aware of the context in given elements so that it can distinguish the meaning of elements according to part of speech tagger (verb, noun, adjective, etc.) in a different context. For example, "incorporated" is a past tense verb and it should stay in a similar context as a verb. Whether staying in the same morphological context or not, the stemming only clears 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, question analysis or named-entity 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, one can call bi-gram modeling for the example as mentioned above 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. Therefore, the questions are *"how will a natural language processing method decide the smallest size that should be windowed on size of natural expressions?"*

N = 1 (Unigram) has 20000 parameters in order to so. Respectively, N = 2 (bigram) has 20000 ^ 2 = 400 million, N=3 (trigram) has 20000 ^ 3 = 8 billion, and N = 4 (four-gram) has 1.6 x 10 ^ 7 [55]. Apparently, the more n-gram model the researcher has, the more complex system that a question answering system need to solve it.

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 [48]. As depicted in (1.1), for source probabilities $w1, w2 \ldots wn$ and the number of n-gram $C(xy)$, language modeling can be estimated with Maximum Likelihood Estimation [48].

$$P(w_n | w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})} \quad 1.1$$

For instance, a sentence like *"I would like to know where the error is."* represents a probabilistic chain that can be calculated with (1.1) as below:

P ("I") x P ("would | I") x P ("like | I would") x P ("to | I would like") x P ("where | I would like to know") x P (the | I would like to know where") x P (error | I would like to know where the") x P (is | I would like to know where the error").

The result of the formula (1.1) will show the most probable value through observed data. 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. The chained probabilistic estimation in statistical natural language processing suffers from counting many possibilities of complex sentences.

On the other hand, the Markov Model [48] 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 a language model. In practice, libraries can assess the corpora through the n-gram model, so that the libraries can produce better results in the statistical natural language processing.

Therefore, the next question is about how to evaluate n-gram language modeling in terms of natural language processing. Extrinsic and intrinsic evaluations are mainly used in the phase of evaluation for language modeling [48]. The extrinsic evaluation stands for end-to-end testing by performing all the system functions over again. For example, if a natural language processing toolkit requires assessing the performance of a language model in a software library, the system may perform multiple times to see the results. However, it takes enormous amounts of 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 common dataset would be divided into a test set and a training set and the test set can evaluate the training set without requiring extra datasets.

Perplexity theory refers to a test set that may tell how well the given model predicts the results. It is a measure of how well a modeled language predicts the next word of an item and the more outcomes, indeed, predict the lower perplexity a natural processing system can get. The lower perplexity denotes a better model. As shown in (1.2), the perplexity shows an inverse probability of a model. At some conditions, the dividend goes to zero in case that a test set could not be matched in a training set. In this circumstance, perplexity cannot be evaluated. Additionally, there is a possibility that a machine learning approach occasionally suffers from the overfitting issue. If a training process has occurred more than average, a system would not give the right results given a test set and behaves as if the model were generalizing every test set.

Consequently, statistical methods established the regarding information that needs to be extracted. They decide how to evaluate the probability of result with perplexity formula. Minimization of perplexity [48] helps statistical techniques such as tagging, parsing, named-entity recognition achieving precise results with their test sets, along with reducing the error of training set (1.2).

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \qquad 1.2$$

Part of Speech Tagger (POS): 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 [48]. Part of Speech Tagger mostly employs a Markov chain algorithm that is a part of statistical natural language understanding. As previously mentioned, the Markov model stands for a state that 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 [48].

Another example could be that prepositions occur before nouns or determiners occurs before nouns, adjectives, and superlatives. In that case, pre-saved corpora which have a million words have to be annotated by POS Taggers. One of the standard lists that has an identifier for POS is named Penn Treebank. A treebank is used for annotating the syntactic and semantic structure of a sentence with million words of part-of-speech tagged text. The selection of a corpus is equally essential to achieve a result with a parsing process.

A concern of *the Penn Treebank* is to provide multiple syntactic bracketing if necessary [56]. Multiple brackets are important for example the *Brown Corpus* tags *"one"* and *"the one"* as Cardinal Numbers but it *"the one"* case could be an important determiner in any sentence. Each tagger is named as labels, which can be at 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 collective noun or an adjective phrase, but those phrases could be identified differently with tagger according to the Markov model of the item in a sentence.

Parsing: POS tagging does not interest in the relationship between tagged elements. A tagged element could coherent with other labeled elements, which is solving with parsing methods. Parsing methods are grouping the tagged elements syntactically, and POS tagger could be though as tokenization method of the parsing process. As a natural language expression 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 to overlap *subject-predicate-object* triples correctly in linked data. The approach of parsing is separated into two main sections, that are, the rule-based approach and the probabilistic approach [57]. 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 rigorously to get the correct answer. The open-domain question answering systems utilize this approach because of the lower complexity of the bottom-up approach and expandable question types. Nevertheless, a rule-based approach could give undesirable results in a restricted domain question answering or semantic question answering and could be time-wasting at parsing.

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, the *"linkedfactory"* keyword might be combined as an adjective "*linked"* and a noun *"factory"*. If the parser went into the deepest leaf node, it would have been relatively faster operation.

Various types of probabilistic parser have prevailed since the natural language processing investigations started. It depends on the grammar of the English language and

how much profoundly information source that is required by a question answering system. Formal Grammars of English define a constituency parse approach, which can identify noun, verb or adjectives in a big chunk like shallow parsing. This approach eliminates item relationship among nouns, verbs, and adjectives by providing an abstraction method. If a question answering system needs a connection 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 [48]. Syntactic parsing suffers from the "*word-sense disambiguity*" problem. This problem denotes that a word can represent different meanings in the sense of its location in a sentence. For instance, "*What does linkedfactory contains*" could be differentiated "*Could you give me the members of a factory which was linked?"*. Both sentences are semantically similar, but they are hard to recognize by lemmatization and sentence similarity methods.

In order to alleviate the *"word sense disambiguation"* problem, an NLP system may use *"bag-of-words"* or *"collocation-based"* concepts. The *"bag-of-words"* is a subset concept of n-gram modeling. Theoretically, every single element is assigned into an array, for instance when comparing the following sentences.

BagOfWords_1 = {"Is":1, "the":1, "system":1, "health":1, "good":1, "for":1, "sensor1":1, "in": 1, "machine1":1} 1.3

BagOfWords_2 = {"show":1, "the": 1, "system":1, "health":1, "status":1, "sensor1":1, "machine1":1} 1.4

Vector_1 = [1, 1, 1, 1, 1, 1, 1, 1, 1] 1.5

Vector_2 = [0, 1, 1, 1, 0, 1, 0, 0, 0] 1.6

(1.5) and (1.6) scored as binary whether a word is the same with its counterpart or not. In most cases, this sort of evaluation is unpractical because semantic structures of words are not taken into consideration except for the frequency of the syntactic structure. In practice, natural language libraries use *"collocation-based"* which the WordNet and the POS-Tagging help the calculation of semantic similarity through the probabilistic Markov Chain.

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

constituency parser is likely known as a phrase parser that has an objective is to check the grammatical structure of sentences by parsing the chunks of the morphological structure. The constituency parser may not handle the relationship among language items. Constituency parser utilizes the Penn Treebank [6] which is used for evaluating as a test set by calculating the perplexity of probabilistic phrase. The dependency parser analyses the grammatical structure of natural input to define the relationship between the root word and the rest of them. Dependency parsers employ algorithms with a dataset of "universal dependencies"[7].

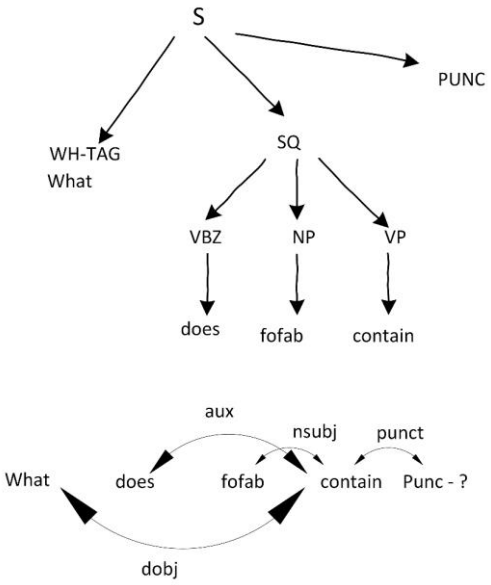

Figure 4.1: A Case Study of the Constituency and Dependency Parser

The constituency parser may cause the disambiguity (see Parsing) problem. The disambiguity can change the tree order in the graph shown above (top graph - Figure 4.1). This disambiguity can be reduced with the use of dependency parser concurrently. The constituency parser can help to realize shallow and deep parser. Regardless of the rule-based or lexicon-based parsing process, a constituency parser extracts the concatenation of phrases.

Shallow parsing extracts (1.7) and (1.8) to define primary phrases with its Penn Treebank assignment. Given a S natural input, each SQ main clause, WH question tag, and $PUNC$

[6] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
[7] https://universaldependencies.org/

punctuation of sentence such as question mark would be extracted to keep going with deep phrase elements (1.9), (2.0), (2.1). A complete example of deep parsing and shallow parsing can be seen in Appendix A.9.

$$S := WH + SQ + PUNC \quad 1.7$$

$$SQ := VBZ + NP + VP \quad 1.8$$

$$VBZ := does \quad 1.9$$

$$NP := fofab \quad 2.0$$

$$VP := contain \quad 2.1$$

As shown in Figure 4.1, a dependency parser focuses on syntactic units between *"head"* and *"dependent"* of which they have composed.

$$contain := punct + nsubj + aux + dobj \quad 2.2$$

$$what := dobj \quad 2.3$$

$$does := aux \quad 2.4$$

$$fofab := nsubj \quad 2.5$$

$$Punct := punct \quad 2.6$$

(2.2) is a *"head"* and (2.3), (2.4), (2.5), and (2.6) shows a *"dependency"* onto the head element. This parsing is a type of semantic parsing. For instance, *"what does fofab contain?"* and *"what contains fofab?"* have similar "NP" and "VP" tag at the same order. Whereas, the dependency parser calculates a new dependency according to *"head"* word, mostly a verb, and assign the other relationships (2.3), (2.4), (2.5), and (2.6) respectively.

Spell Checking and Abbreviation Correction: Spell checker might be an evaluation criterion for the restricted question answering system. It is not necessary to provide an advanced spell checker that controls all aspect of morphological, semantical and syntactical; rather the researcher prefers a simple checker.

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 domains mainly affect the open-domain question answering system because natural expressions should be handled with a variety of questions corresponding domains. 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 abbreviations than a medical field. In this case, the best way to classify an acronym accurately is by using a simple look-up dictionary or hash table. The *Bayes Theorem* and *Levenshtein Distance Algorithm* would be useful to find both spell correction and abbreviation checker. However, the spell correction gives better results than the acronym checker does under the Bayes Algorithm.

Named-Entity Recognition: It is a subtask of information extraction to locate any distinctively named entities with pre-classified labels such as names of people, organizations, locations, quantities, and so on. Named-entity recognition is a method that identifies the item of a sentence as domain-specific. It defines all structures mainly as a person, a location, an organization, and an entity. As shown in (2.7), *"sensor1"* and *"machine1"* are labeled as an entity and are found to have a relation between each other.

$$What\ is\ the\ value\ of\ sensor1^{entity} \leftrightarrow \{subject\}\ in^{relation}\ \{object\} \leftrightarrow machine1^{entity} \quad 2.7$$

This evidence shows us named-entity recognition is an application-specific task. A 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, statistical methods or a rule-based model can create a 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 comprises a combination of *"HeatMeter"* or *"HeatingWater"*, which it can assert given items with the character started by *"Heat"* in a smart factory. The named entity recognition is a higher-level extraction tool that uses statistical methods such as POS tagging, constituency parser, and dependency parsing without the need to do distinct steps.

Word Vectors (Word2Vec and Glove Data): Vector space can represent the similarity context of words. A language system can create the meaning of a word table by converting a word into a vector. For instance, if it is taking into consideration 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 are similar

in terms of frequency matrix of their meanings. *"connected"* and *"network"* are equal semantically. Therefore, one can create a word vector from corpora 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 effective against the non-existent word in a vocabulary. Another issue comes out with *Sentiment Analysis with Word Vectors*. Due to the closeness of question phrases and negative phrases in some context, those phrases make complicate the positive, negative or neutral sentence analysis. As a consequence, the semantic similarity calculates a semantical representation in word vectors.

Sentence Similarity: Sentence similarity is used for comparing two string inputs 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 words, the *n-gram model* or more specifically the *bag-of-words* concept can be implemented. The sentence similarity is closely relevant to n-gram modeling, and it connects the sentence to respective one. The difference sentence similarity and WordNet similarity is that the sentence similarity calculates syntactic similarity takes into account occurring word order; however, the WordNet similarity is the semantic similarity which calculates semantic distance between two given inputs.

Jaccard Similarity: This algorithm uses a procedure to calculate the similarity between sets of data defined as the size of intersection divided by the size of a union of two sets [58]. This similarity method is a type of syntactic similarity.

$$j(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \qquad 2.8$$

Jaro-Winkler Similarity: This algorithm calculates transposition of matrix t, the number of common characters c, and given strings $s1\ and\ s2$ by inserting into a formula as below [59]:

$$sim_{jaro}(s1,s2) = \frac{1}{3}\left(\frac{c}{|s1|} + \frac{c}{|s2|} + \frac{c-t}{c}\right) \qquad 2.9$$

$$sim_{winkler}(s1,s2) = sim_{jaro}(s1,s2) + \frac{s}{10}\left(1.0 - sim_{jaro}(s1,s2)\right) \qquad 3.0$$

The Winkler algorithm increases the Jaro similarity by employing initial characters and gives a similarity measurement [59]. For example, Jaro-Winkler takes head characters of

a string such as *"health"* and *"heal"* to perform the Winkler formula. However, this similarity method is also a type of syntactic similarity.

Levenshtein Similarity: The Levenshtein algorithm has a variety of application areas such as spell checking, acronym finder or sentence similarity. This algorithm calculates the cosine distance of given two strings $s1, s2$ that are divided by the maximum absolute value given two strings [59].

$$sim_{ld}(s1, s2) = 1.0 - \frac{dist_{ld}(s1, s2)}{max(|s1|, |s2|)} \quad 3.1$$

WordNet Analysis: WordNet is one of the largest databases for English lexicon that can be used for word and sentence similarity analysis. Depending 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. However, the significant difference is that the WordNet calculates semantic similarity through a large-lexicon database.

Wu-Palmer Similarity (wup_similarity) [60]: This measure calculates relatedness by considering the depths of the two synsets d (a set of synonyms of a particular word) in the WordNet taxonomies, along with the depth of the Least Common Subsumer $L_p \; and \; L_q$ [61]. With the following formula as shown in (3.2),

$$\delta_{wu\_palmer}(c_p, c_q) = \frac{2d}{L_p + L_q + 2_d} \quad 3.2$$

After defining *Least Common Subsumer*, it is turned out to be a tree-based semantic relatedness measure extracted from the *"is-a"* relationship of a tree. For example, *"contain"* and *"incorporate"* synsets are identical according to the Wu Palmer algorithm. Next, the WordNet finds the first Tree with categories as shown in Listing 4.1 [61] :

```
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) Depth1 = min(depth({tree in T1 | tree contains LCS} )) = 3
7) Score = 2 * Depth of Least Common Subsumer / (Depth1 +
   Depth2) = 2 * 1 / (3 + 3) = 0.3333333333
```

Listing 4.1: Wu Palmer Sample Calculation [61]

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 us embrace the same example comparing to *"contain"* with *"incorporate"* as shown in Listing 4.2:

```
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 4.2: Leacock-Chodorow Sample Calculation [61]

Question Classification is a module, regardless of a domain type, that 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 the derivation of an expected answer.

$$< q_i, c > \in Q, C \tag{3.3}$$

$$C = \{c_1, c_2, c_n\} \tag{3.4}$$

Given a question set q_i and label set c_n, the classification method attempts to find a match between questions and labeled set (see 3.3 and 3.4). If the classification method could find the match, then it assigns a label onto a particular question. The question classification can classify questions with rule-based or machine-learning methods. In reality, the rule- based methods take time for creation, the overhead of manually handwritten rules, and extra processing for case-dependent classification.

The *Logistic Regression* and *Support Vector Machine* methods have been tried to apply for question classification and multinomial logistic regression can assign to multiple class from given questions. 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.

4.4 Chapter Discussion

The Natural Language Processing has two fundamental problems, which are the *"word-sense-disambiguation"* and *"out-of-vocabulary"* problems. *"Out-of-vocabulary"* linguistic elements such as words are not recognized while testing with natural language processing system due to non-existence vocabulary. Special words might have encountered in regarding *"out-of-vocabulary"*. In particular, the knowledge-based question answering generally suffers from *"word-sense-disambiguation"*. This problem complicates stages, which the researcher has defined in Section 4.3. Data preparation and the utilization of semantic technologies are equally important as the researcher selected methods for the Semantic QA.

The language modeling and perplexity are more theoretically oriented methods to represent a language as series of strings. To create a statistical model, they need to follow rules of language modeling, which should expose error evaluation with the perplexity and probabilistic estimation with the maximum likelihood formula for language modeling. The rest of them are strictly relevant to practical usage such as dependency-constituency parsing, POS tagger, named-entity recognizer, stemming-lemmatization. In accordance with H-2 and H-3, rapid results and precisions are dependent on combinations of the methods that the QA used. Reinforcement learning and deep learning methods, such as LSTM, for general QA needs a large amount of data and training time in reducing training error with epochs. Principally, our methodology is making a question analysis as deep as possible to extract mapping features for an ontology pattern.

Semantic web technologies become usable when the point comes to the QA. Most of the QA applications are document-oriented; nevertheless, some QA applications try to map the ontology source to patterned questions, which is an exactly opposite approach compared to the methods described in this thesis. The SPARQL language is still hard to be grasped by human operators. It appears to be the most efficient way to work with linked data. Hence, the template-based SPARQL language can reduce operation time and query complexity by increasing the efficiency of human operators. When a QA application aims to utilize a heterogeneous data source, the complexity of underlying algorithms usually becomes higher. In spite of the higher data size than regular ontologies or documents, the predicates of semantic in streaming data source have more data that have not a meaningful set of predicates. Lastly, RQ-3 and H-2 can be actualized through semantic technologies and heterogeneous data sources.

5 Practical Implementation

The practical implementation was realized with *"ASP.NET Core"*, *"Angular 2+"*, *"Flask MicroFramework"* and a template language named *"Jinja"*. This frameworks and template language will be enlightened in the following sections. The *ASP.NET Core* and *Flask Microframework* were acted as a solution to the back-end development; the *"Angular 2+"* 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 Architectural Design in the Operator Assistant Web-Based Software

REST stands for the *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 the microservice architecture pattern has been used, the selection of architectural pattern varies from the 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 protocols, such as SOAP and complexity operations requiring addressing and content to be arranged over again. As shown in Figure 5.1, the web-based software relies on three primary components that deploy separately. For instance, blue and gray-colored components can be deployed independently, and there is no dependency between them. The web-based software can decouple the orange and gray-colored components, which are the *OPC UA Web* and the *Semantic QA Components* respectively, without changing any addressing, content or negotiation protocol. After adding a load balancer that 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 the *eniLINK* and natural

language server toolkits. The test page in the orange rectangle in Figure 5.1 has no login authentication, but it has a JWT authentication module that can verify the private token of HTTP requests. If this test page would have been installed independently, it does not harm the security of the software. Nevertheless, a developer can add a login-authentication to that module, or a transport layer security can be added such as HTTP Secure via a common load balancer.

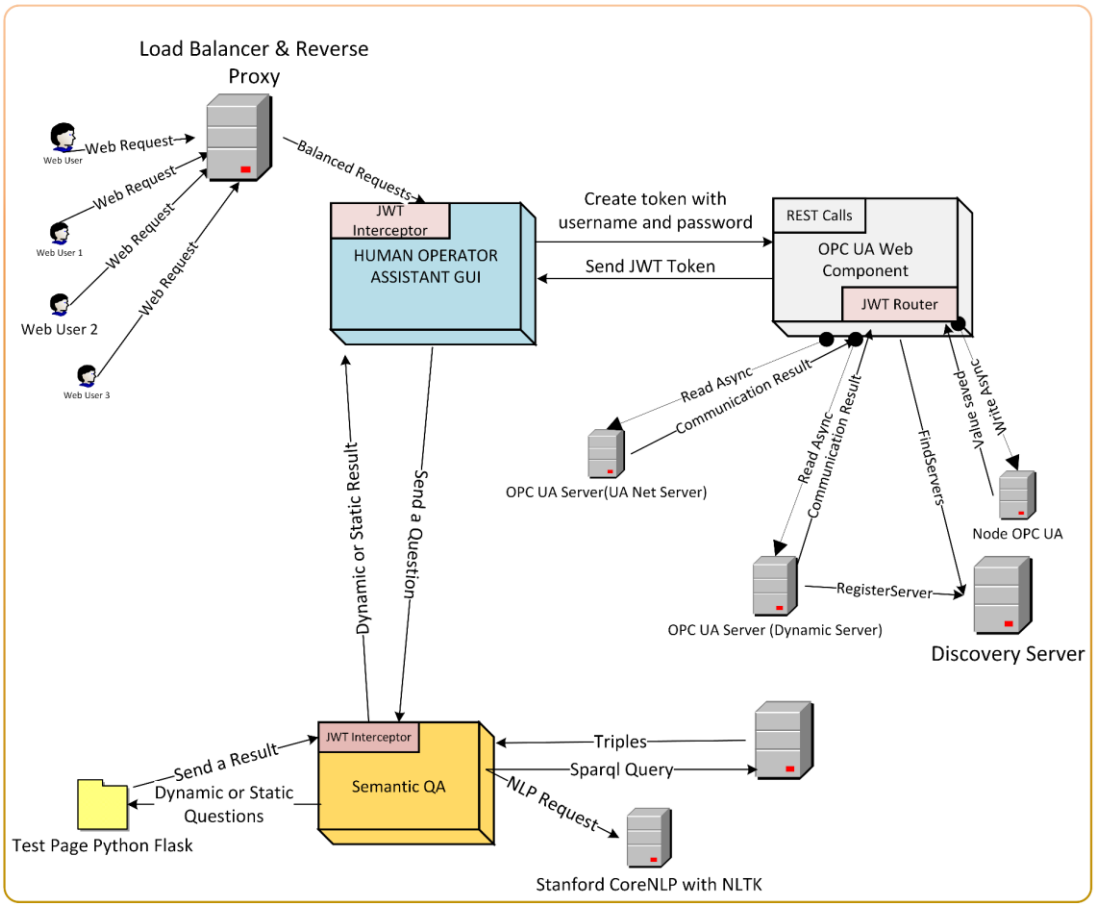

Figure 5.1: RESTful Architectural Design of the Web-Based Software

ASP.NET Core commences the routing entry point with *"API"* keyword. *"/api"* is a mandatory routing field to be sent a set of HTTP requests. For the first time, the body of a request should carry a username-password pair to initiate the token-based authentication. HTTP method should be the GET method because this method has a characteristic called idempotency. HTTP requests work the way of pipelining request to reduce transaction numbers between request and response. Although this feature boosted the quality

of request-response communication, the *"POST"* method may get in danger; hence multiple requests of the *"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. HTTP Caching can reduce redundant data transfer and round trip time by saving local storage instead of storing into servers. The drawback of the caching is that a response can be cached and displayed to an unauthenticated user. Another disadvantage the data content may remain the same whatever a user has made a request.

The *OPC UA Web Component* holds username-password pairs in the way of embedding into the application. Hence, the body of requests will take the username-password pair for authentication token as established as depicted 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 a server which defined in *"appsettings.json"*. The researcher has saved hard-coded server lists, but the researcher can also obtain from a discovery service in the form of a string array (see Appendix B.16).

```
[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 to protect from vulnerable attacks. However, simulated data in any object can allow writing data in the system for the testing purpose. The *"message"* can be a primitive or user-defined type, and it depends 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"*.

```
[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 easy integration to the OPC UA Web Component. As depicted in Listing 5.4 and Listing 5.5, separation of the message types are important to realize while considering the Semantic QA. Having inseparable semantic representation between queries about which a request is concerning streaming data or static data, the web-based software should apply multiple HTTP requests as shown in Listing 5.4 and Listing 5.5.

```
[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

Subscription requests consist of a monitor id to create a polling mechanism. As mentioned in Listing 5.6, subscription requests need to set monitor id arrays. The web-based software utilizes a POST request to update *"monitor_id"*. A sample preview can be found in Appendix B.15.

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

Listing 5.6: HTTP Post Request for Monitoring Node

HTTP Requests can be repetitive methods triggering by a user. Taking into consideration an increasing amount of data in industrial networks on a daily basis, the researcher should design an architecture that complies with the load balancing and non-blocking input-output queues. A load balancer can balance workloads by using the predefined algorithms. In this work, the researcher will use the round-robin algorithm and the least connection algorithm. The round-robin scheduling chooses guidance for workloads by putting into queues according to predefined weight. The least connection takes care workloads, which takes a different amount of time, and it specifies that the fewest loaded queue in order to avoid stall of the workloads.

5.2 Front-End Development

5.2.1 Overview

Angular 2+: "*Angular 2+"* is an extension framework that was developed various properties of *"AngularJS"*. Libraries of *"Angular 2+"* are not suitable to work legacy usage so that the *"Angular.JS"* Framework cannot use any library from "Angular 2+". The main reason for the underlying framework has been written in the *"TypeScript"*, not in the *"JavaScript"* library. *The Angular Frameworks* organize the dependencies of modules with *"package.json"* which is a command line feature. All sort of libraries is saved into *"package.json"* to detect discrepancies between versions of libraries. *"Angular 2+"* allows embedding dynamic bootstrapping features into a pure HTML Page. The biggest drawback of Angular 2+ is that it is not backward compatible, and the differences between versions can be immense. While *"AngularJS"* follows the pattern of MVC, *"Angular 2+"* implements a component-based pattern. Besides, *"Angular 2+"* splits component by component to increase code reusability and achieve the object-oriented paradigm in script languages. *"Angular 2+"* is faster than *"AngularJS"* versions because *"Angular 2+"* uses different hierarchical dependency for each module. When a developer updated a module, only regarding hierarchy is 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 the *"Angular 2+"* provides an internal library implementing an asynchronous feature. *The Angular Frameworks* utilize two-way data binding that allows performing data flows in two directions co-occurrence.

Ember.Js: *"Ember.Js"* is a JavaScript MVC Framework that helps to organize large web applications. The structure of Ember.js depends on micro-libraries [62]. MVC pattern fully complies with the *"Ember.Js"* in terms of bindings, computed properties and automatically updated templates [62]. Bindings enable the change of a variable propagating to another variable. Using calculated *Properties* and *Automatically Updated Templates* ensure the framework stay up to date with regarding data source of Ember.js. One of the significant advantages of *"Ember.js"* that is the data library of *Ember.js* which stores all values of a process employing caching into the *In-Browser Store* [62]. Ember.js supports all end-to-end testing tools, such as *Karma* and *Mocha*. Testability is an important step to develop bug-free codes so that one can conclude that Ember.js has a variety of compliance with test tools.

The primary purpose of *"Ember.js"* is to support the single page application (see Appendix B.1); thus it has no architectural layer for server-side rendering. The server-side rendering is an old transfer technology for HTML websites and brings a significant overhead in case of minor changes. Moreover, *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. The *"Ember.Js"* is fully backward compatible that means one can use a function from an old version in a new version.

React: The React library serves a useful purpose to make a full-viewer of a front-end library. The React is primarily concerned with the view aspect of the user interaction, and it is not suitable to use as a framework or library in a large-scale application [63]. React does not enlarge 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. Another drawback is that the React supports one-way binding, which updated UI elements do not propagate to the *"View"* immediately. React has been posited that front-end developers can leverage its features in creating the part of a viewer in MVC. In essence, the React follow does slightly follow the MVC Pattern. This is an advantage over other frameworks because everything is a component even if it is not used. The React Framework can decouple as compared with *"Angular 2+"*, each component uses its view and regarding data. The philosophy of the React divide the components as small as possible. Lastly, the React library created a breakthrough step in utilizing *"virtual DOM"*. Such HTML elements and stylesheets are updating in a stateless way entirely when a request has occurred. In *"virtual DOM"* technology, the DOM elements are updated through an intermediary DOM refresher regarding affected parts of the React elements. In this way, it could reduce updating time and increase the speed of user interface compiling.

Meteor: *"Meteor"* 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 [64]. The underlying structure is based on Node.js and its virtual box named *Google V8 Engine*. The underlying mechanism of *"Meteor"* detects the changes of the object and automatically set the results before a developer made. Angular2 and React have observables to ensure sort of properties.

Vue.js: *"Vue.js"* is a frontend framework that has a similar grammatical structure to *"React"* and *"Angular.js"*. Templates are one of the powerful features that are used by Vue.js.

Vue.js provides data bindings through templates. Templates support two-way data binding, that means when you changed an input, the Vue.js will update the corresponding element. After combining VueJS elements with HTML, every element of Vue.js will be reactive, which inputs are rendered forthwith accordingly. Vue.js is a component-based system that the abstraction mechanism of language works with components. Methods can be executed in Vue.js through cached memory. Thus, a cached method is not compiled in multiple requests so that a Vue.js application can reduce the memory complexity of method calls.

The main characteristics of front-end frameworks and the underlying script languages can be found in Appendix B.8 and Appendix B.4 respectively.

5.2.2 Implementation of Front-End Development

"Angular 2+" applications have component-based design and object-oriented script languages, DOM elements of components and component export file. The 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 2+*, 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 2+. Subscribers put HTTP requests into the event handler to send a request. This request value would be saved payload, event handler trigger, and an error trigger. Whether server-side had occurred any error, client application shows the result with *"HttpErrorResponse"* inside of constructors that holds errors, appropriate headers, status, and URL information.

Components represent classes of Angular 2+ that connects DOM elements such as ".html" files in connectıng with data binding. Each component has a selector and template URL to attach 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 2+"* 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 the application and makes the unit test possible to inter-class dependency.

Document Object Model (DOM) manages markup languages and stylesheet documents that organize the display format of markup languages. DOM elements and object-oriented script files are in a robust two-way data binding relationship. Data binding is an essential feature in *"Angular 2+"*, hence 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 statements that *"Angular 2+"* used. 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 tokens*. JSON Web Token may provide a stateless and stateful connection across all incoming and outgoing requests. In such circumstances, 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 of the front-end and back-end side.

In order to improve the 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, the microservice framework (Appendix B.1) would be possible for future use at a different facility of a smart factory. Question Answering leverages a template-based front-end application, as well as the DOM elements, are generated in the same back-end framework. However, multi-threaded Flask applications strongly apply the integrity of HTTP requests without delay and communication failure. Another advantage is that there is no third party library requirements for data-binding and performance evaluation of the semantic QA. Consequently, single page applications can handle data the way of dynamic binding and single time loading of DOM elements from different back-end applications that are used with different programming languages.

5.3 Back-End Development

5.3.1 Overview

The following statements have a brief description of the back-end development process in terms of a framework that was implemented in the experimental development of OPC UA, the *"Information Model Mapping"* for Semantic Data and Semantic QA. All of these development cycles are examined with a comparison between frameworks, languages, libraries, and toolkits. Regarding the *OPC UA Web Component*, frameworks, languages, and software toolkits are taken into account.

ASP.Net Framework (Active Server Pages .NET): *ASP .NET Framework* is one of the oldest frameworks that is utilized by developers to implement web applications. The oldest framework named as *"ASP.NET Web Forms"*. The ASP.NET web forms were strongly dependent on the Windows Operating System because of the *"Internet Information Service"* (IIS). This server deploys 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 known for *Web.dll* [65]. The *Web Forms* evolved to ASP.NET MVC Framework to comply with the *Model-View-Controller* design pattern.

ASP.NET Core: In addition to the continuous improvement of this technology, Microsoft decided to scale this framework in Unix-based architecture. Therefore, the name of the technology was changed as *"ASP.NET Core"*, which brings the developer worlds lightweight features. One of the most prominent features of *ASP.NET Core* is the advanced routing framework to control RESTful API calls. When an HTTP call arrives, it should be parsed as a schema and host path. Schema path decides on which the transfer protocol took 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 is used for a testing environment because of the immature step of development like ASP.NET Framework. 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.

Node.js: It is a framework based on JavaScript language that leverages a virtual machine developed by Google Inc. [8] in order to achieve an event-driven server-side development framework. By leveraging a virtual machine named *V8 Engine*, the framework sends an event signal to the virtual machine rather than communicating an operating system itself. Node.js has extended support for multiple operating systems because the virtual machine has been compiled for numerous 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 formerly named as *Native Call* in the software world that is very useful when a function result returned from a programming language managed in a virtual machine with a native language. Asynchronous callbacks implemented a finite state machine in C++ language unless 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, garbage collection which based on the virtual machine, routing and dependency injection. It has an object mapper such as *Entity Framework in ASP.NET Core* that has an ability to connect with databases mapping primitive objects 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. The *Spring Framework* works with *Plain Java Objects(POJO)* to simplify for the purpose of programming and testing modules.

Flask Micro-framework: Many software developers take advantage of rapid prototyping by applying the Flask micro-framework. For the reason that the researcher needs to develop a rapid-prototyped solution, Flask helps developers in many aspects. The Flask works multiple versions of Python 2.x and 3.x. There is no complicated routing mechanism, and it is utterly compatible with microservice development. Annotators are bounded and compact, which they make 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, the researcher faced the biggest problem of Flask was the non-asynchronous communication pattern. One of the proposals of the researcher is to remedy the problem that the system can use a non-blocking input-output queue with an asynchronous task queue. A non-blocking input-output queue can store convenient packages by increasing performance of all requests in a queue before they reached to the

[8] https://www.w3schools.com/nodejs/nodejs_intro.asp

application. Broadly speaking, it is far more than making a routine asynchronous. When an asynchronous queue works, it selects a message broker to connect with 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 the Flask application does not have the *"async"* keyword that makes the routines asynchronous call.

Django Web Framework: It is a loosely coupled, high-level Python Web framework along with supporting *Model-View-Controller* pattern. It takes benefits the language property of Python allowing indented programming and implicit data types. The 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 in the architecture of the framework, and predominantly it is associated with versions of Python 2.x and 3.x causes discrepancies among libraries. Regular expressions with precedence rules supply the routing mechanism of Django. When a URL is matched, all other requests are dropped per precedence rules. The Django Web Framework can consolidate URL Patterns by including a URL one into another. In this way, developers can efficiently manage the URLs and HTTP Requests within a single base URL entry point.

Likewise, the latency is a parameter that indicates the duration of the delay within consecutive queries. For the purpose of testing, each request is processed to fetch a single row from a database, and the data is serialized as JSON [66]. Hence, low latency gives better performance for back-end frameworks.

5.3.2 Authentication of the Web-Based Software

A web-based software must have been in compliance with an authentication standard anyhow. Principally, there are two kinds of authentication, which can be observed in the web-based software, which are certificate-based authentication and token-based authentication for end-to-end security. The end-to-end security emphasizes that the communi-

cation 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 inconvenience to obtain data by a malicious person without a private key.

On the communication level, a web application can handle the authentication issues as well. OPC UA clients 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. These two communication models commonly need to send initialization parameters such as Acknowledgement(*ACK)*, Syn Flag(*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 a more robust design than HTTPs and TLS, but support standardization among OPC UA Servers is very low. For example, [Cavalieri, Salafia & Scroppo 2018] [7] [6] [36] has an implementation is ensuring the WebSocket connection with a particular technology which is designed by Microsoft [9]. This approach solves the standardization problem for the WebSocket protocol; therefore, the web-based software only supports connecting through 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 the 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 a client wants to authorize while a process is executing to refer the token ID, the name of the issuer and issued time as Unix epoch date. The last part is the verification to signatures, 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 a 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. The *"SecurityTokenDescriptor"* defines expiration date and type of credentials such as *Aes128*, *HmacSha384* (Symmetric Encryption) or *RsaSha256Signature*

[9] https://github.com/SignalR/SignalR

(Asymmetric Encryption). In addition, the practical implementation of a symmetric key, which has a "Hmac Sha1-256", cipher.

A compact way to provide security is to implement an authentication method within the low-level protocol area. Catering to protocol-level security, the OPC UA Web Component employs a certificate-based authentication through an X509 Certificate to a certification routing as illustrated in Figure 5.2. This routing system prepares an authentication initiative to decrypt parameters of a certificate. At the same time, the OPC UA Web Component sends a secret message through the Open Secure Channel which is initiated by a Session, and message-certificate item pairs are verified with an asymmetric signature.

JWT Authentication module of the OPC UA component covers the Semantic QA component. Instead of creating multiple secret keys, the web-based software creates a single key to confirm the data integrity of the main component that 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 distributed for each components easily, and this does not harm the design principles and performance.

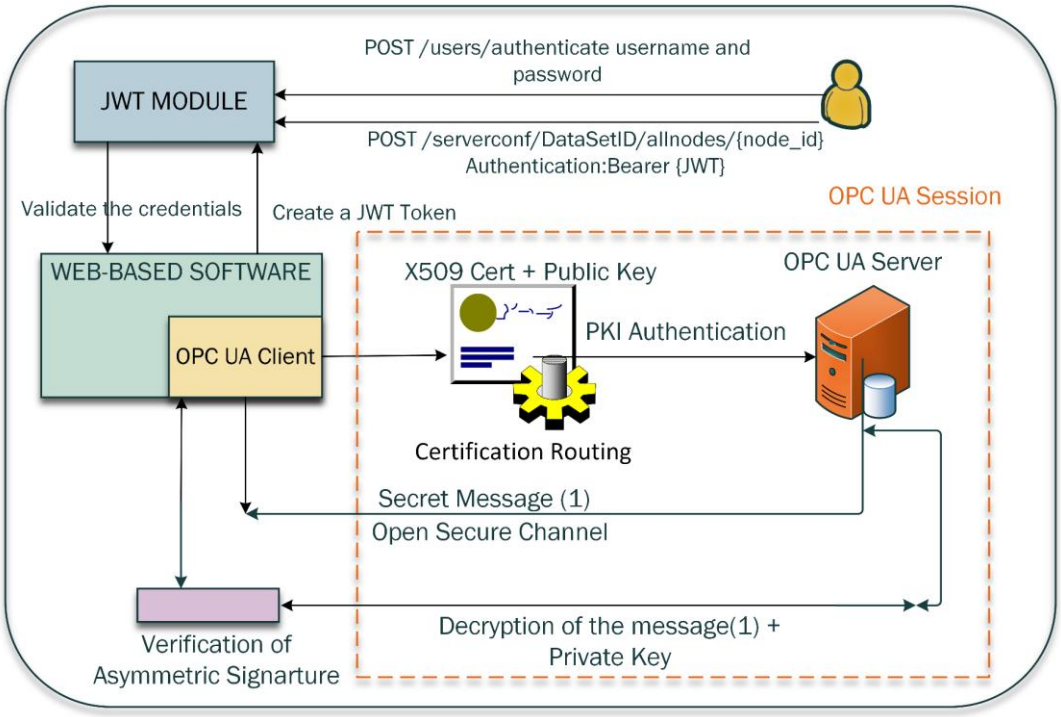

Figure 5.2: Authentication System in the Practical Implementation

Lastly, JWT Authentication is not the only method that a system can utilize through an authentication concept. *Kerberos* and *OAuth* could be intended to use instead of JWT Token for the purpose of the authentication concept. In common, all three authentications employ issued tokens, but there are some slight differences from each other. JWT is an unloaded authentication, which means that an authentication system interrogates tokens in incoming requests. A request that consists of JWT Token does not require extra configuration against endpoints. However, the *Kerberos Authentication* needs an essential distribution service and an authentication server. Initially, an incoming request is sent to the *Authentication Server* to authenticate the request by user name and password pairs. If a symmetric key is used for authentication, the key distribution server should handle all requests to exchange keys instead of the public-key infrastructure. This could lead to the point of failure and security vulnerabilities. Another drawback about the *Kerberos*, time limitation of a ticket does not allow time-to-live value such as JWT and OAuth and this time restriction must be synchronized with clocks between servers. Consequently, the JWT authentication is one of the most powerful authentication methods that can be implemented without the needs of extra overhead, configuration, and software layer.

5.3.3 Connection Establishment of OPC UA

OPC UA Connection has been achieved with .NET OPC UA Standard SDK [67]. A session needs 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"*. The 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, clients and servers should match their built-in certificates to be exchanged. A server behaves against incoming client requests as untrusted communication at the initial stage. Thus, manually accepting certificates could be a necessary intervention for the sake of compact development. To eliminate this kind of intervention, a global discovery server may provide a satisfactory approach. All OPC UA servers in the same network trust the discovery services as they started to send notification on which they deployed. 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 may be polled by reading a particular node value that shows server status or monitoring node concept. Monitoring nodes can support active polling within a smaller time interval rather than reading node values with consecutive queries. However, this approach creates extra overhead for client-side applications. A reasonable way to handle that this is using a communication stack function of an SDK that polls to server status under a structured data type such as *"DataValue"*.

"What if the web-based software has been integrated into a different division of a smart factory, how would the central part manage certificates?". This question caused to be unavoidable fact implementing a *Local Discovery Server* and *Global Discovery Server* as mentioned previously in Section 3.2.4. A couple of practical installations for the discovery services have been given with the main features in Appendix B.14.

The reading, writing, browsing through nodes, value or attributes utilizes an accurate asynchronous session notification. This notification put the session information into the *"mutex"* (thread-safe locking method) to ensure another request would not use the same session. Servers can fail incoming requests under the circumstance that a server considers security or connection parameters of clients have not complied with default parameters. Then, a client does not need to restore a session; the web-based software should only open new sessions to connect servers. Nevertheless, this could lead the client to deteriorate the 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

The *"Data Navigation"* is used to read, browse, and write values into an *"OPC UA Server"* as being a part of the *OPC UA Information Model* and *Address Space*. The serialization and type checker have been taken from the implementation of previous studies and they have contributed to the thesis which has published by [6] [7] and *Free OPCUA* [51] in terms of built-in typed data serialization and JSON Data Serialization. The *"Data Navigation"* represents the background behind the reading, browsing, and writing requests of the *OPC UA Web Component*. The web-based software applies software development kits procedures to navigate and polling data through *"OPC UA .Net Standard"*[10] library.

[10] https://github.com/OPCFoundation/UA-.NETStandard

Besides, the web-based software employs the serialization methods in *"Newtonsoft.JSON.Linq"* [8].

The idea behind of serialization is converting from the built-in types of an OPC UA Server into JSON schemas or values. By way of JSON serialization, the web-based software can use the information through *"HTTP Request Payload"* to exchange data between front-end and back-end architectures. Even though the features of built-in data such as description, binary schema, or field type are saved as an XML Schema in OPC UA Protocol, the XML format is not suitable for neither OPC UA web-based software interaction among modules nor for a semantic question answering. The approach comprises *"Built-In"* data type which is converting into a JSON data through serialization to send a proper response from OPC UA Servers [7] [36]. To navigate between *"Built-In Types"*, XPath navigation is exploited that includes over 200 built-in functions for string values, numeric values, boolean values and node manipulations [68]. The web-based software deals with "user-defined datatypes" might be created from *"Built-In Types"* to ensure platform-dependent types on primitive types. Serialized JSON data should be compliant to a schema definition of JSON [7]. JSON Schema has a performance advantage over the XML Schema during extraction items from a schema. In addition, it has a similar structure to semantic representation more than XML Schema. The most significant disadvantage is that the JSON Schema takes time in validating nested-data structures longer than the XML Schema.

Most of the 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 Schemas. In spite of being a structured data type, information parsing are relatively comfortable with labels within an XML Schema. The *"Type Checker"* is responsible for controlling node objects and attribute types. After loading the type definitions, the type checker ensures the compliant type information of attributes and node classes to be read and extracts elements of "*OPC UA Protocol Stack"* into JSON Schema.

The *"Data Polling"* mechanism of the web-based software 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 intervals. In the practical implementation, a developer 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 limitation restricts to follow changes of manufacturing devices, and

the only solution could be redesigning OPC UA Server in order to subscript changes within a particular time interval. Subscription connects to an existing session to prevent creating an excessive number of sessions. A lived session can be controlled from a shared pool that has all session's identification numbers with their general configuration.

After subscribing a request, the software 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 even though a user or an internal application can make the selection. For instance, monitored part of the application may select 1000 milliseconds sampling interval, which is the most common interval rate selected by OPC UA Servers. If the 1000 milliseconds were selected although a server did not support the rate, the server assigns the most applicable rate to apply a sampling rate. The selection process could be different server by server. Regardless of a sampling rate below than a particular (default) value or upper than that, a client creates a subscription to insert monitored nodes into this subscription against an OPC UA server. A user can take an exception from the protocol stack of OPC UA regarding mismatched interval rate; either way, a subscription is produced. Another finding is that the underlying structure of the subscription mechanism of OPC UA Servers is not suitable in the context of multi-threading. It is rather evident that client-side implementation should be aware of cyclic delays as well as multiple monitoring nodes could create a delay for results.

5.3.5 Implementation of the Semantic QA

The approach of practical semantic QA was heuristic-based syntactic analysis, and each question should follow the steps of query formulation phase in the flowchart as illustrated in Figure 5.3. The semantic QA concentrates a static query and dynamic query, which a static query infers questions that can be asked against the QA in terms of static data source such as eniLINK data and OPC UA generated data. Unlike a static query, a dynamic query is a type of questions that can be inquired against the QA in terms of streaming data through the *"KVIN"* Service.

An input should clean additional characters of a string with the removal of the stop word and tokenization of an input. All of the elements illustrated in Figure 5.3 have represented in different classes in an application. The stemming erases the prefixes and suffixes of verbs to compare the synonym of the word. The lemmatization turns given verbs

into a concise form by eliminating the English tenses. Unlike an open-domain question answering, the researcher could not implement a rule-based approach. For instance, [Unger Et al. 2012] proposed a template-based approach that gives accurate answers with a predefined set of a statement such as *"<noun, property, object>"* into template SPARQL.

Theoretically, heuristic-based syntactic analysis is one of the closest solutions for this study to achieve proper results. In practice, template-based or pattern-based approaches decrease ease of use and expandability; for that matter, the QA may affect the serviceability of a human operator. Machine learning based algorithms need a high sample size and extended data coverage. The reason that the semantic QA used a heuristic approach is able to get an answer within less amount of time with a limited data source. Heuristic-based syntactic parsing does not search for all possible states except that it begins with a most possible state. For example, the semantic QA has a couple of control state on the *WordNet Verb Analysis*, the lemmatization, the dependency parsing, and the named-entity recognition and the QA makes an effort to find out possible solution states.

In other respects, if a question answering system works with a set of collected documents or labeled ontology data, the system can score answers of questions to indicate in a better way. In reality, the answer selection, which provides the most convenient answers out of the collected answer set, is not suitable for the semantic question answering in a restricted-domain according to the experimental development. Data scarcity does not only affect implementing a machine-learning algorithm, but it also makes a question answering, which based on information extraction, harder in realization. Accordingly, a semantic question answering that exploits restricted domain data should focus on an in-depth parsing approach. After implementing the constituency and dependency parsing, a question classification is used for eliminating answer types rather than scoring the answers.

The number of steps in the semantic QA can be reduced as shown in Figure 5.3, which this alteration may increase answer return speed for the performance of the semantic question answering. As illustrated in Figure 5.3, each color represents a different process of the QA. While the parsing is a syntactic finder as shown green colored, due to the specialization process, the *Question Classification* and *Query Formulation* steps are displayed in a different color as illustrated orange and red colored. The rest of the process is a normalization process; namely, it clears unnecessary parts of a natural language statement in the context of feature engineering.

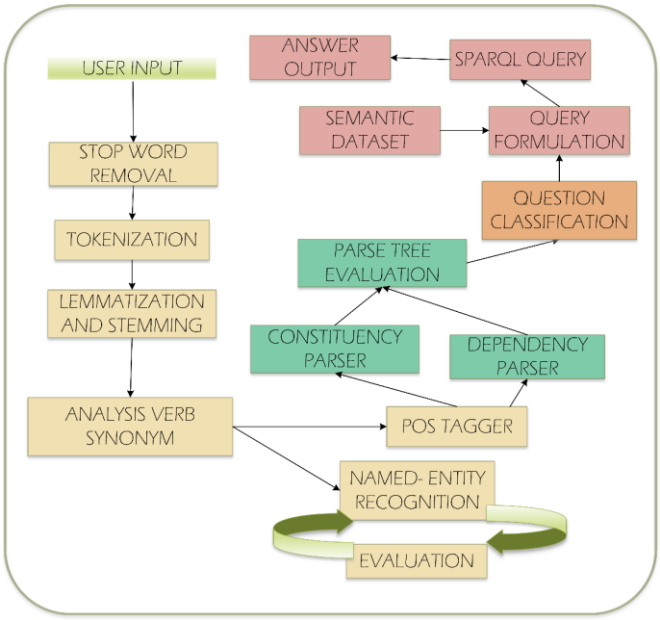

Figure 5.3: The Flowchart of the Semantic Question Answering

If statements, i.e. *"What does linkedfactory contain?"* or *"Please give me the linkedfactory's members"*, were asked to the question answering. The constituency parser realizes deep and shallow parsing to find out the syntax of linguistic elements. A sample result of deep and shallow parsing can be found in Appendix A.9. As explained in Section 4.3, parsed elements may reserve disambiguation. Notwithstanding, the disambiguation does not break grammatical correction in given natural language expression. In this way, the semantic QA can confirm the grammatical corrections of a statement. A question can change predicate, object and subject orders according to requirements of a human operator. In this case, the dependency parser detects the syntactic relationship between predicates, objects, and subjects. Indirect dependencies have been created for the static query such scenarios *"What does fofab contains"* and *"Which one contains gmxspanen1"*. Thereby, two different hierarchical results come out the existence. The dependency parsing can find a root verb relationship such dependencies as depicted in Figure 5.4. Hence, the dependency parser can identify the subject or object relationship between constituent for recognizing indirect dependency.

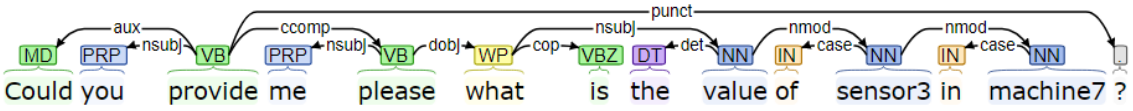

Figure 5.4: Dependency Parser

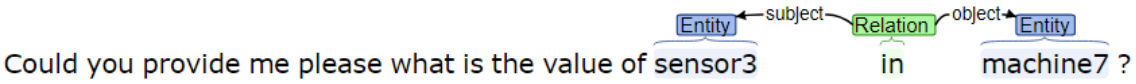

Figure 5.5: Named-Entity Recognition

The semantic QA calculates the semantic similarity for verbs of a static query. When the semantic QA encountered a matchup between a given word and synonym verb through the WordNet, the query process is then able to keep going until it gets an answer. Unlike the semantic similarity, the QA calculates the syntactic similarity to a simulation environment for given dynamic queries. For example, *"Is the system health good for sensor1 in machine1?"* can show error depending on the outcome of operation that a human operator does. Typically, the dataset has not composed of error-label properties. However, the web-based software may set an error threshold value for dynamic queries. Thus, a human operator can ask a question that is syntactically similar to the question as mentioned earlier. (2.8), (2.9), (3.0), (3.1) and (3.2) can calculate the syntactic distance among multiple strings relatively.

The *Query Formulation* has to map syntactic elements onto SPARQL query patterns. For a static query, a response from the SPARQL endpoint as shown in Appendix A.10. For the dynamic query, the algorithm as shown in Figure 5.6 evaluates the queries in 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 semantical separation between static queries or dynamic queries. Regex-based rules are used to discriminate keywords as mentioned above.

Algorithm 2 Query Formulation

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

Figure 5.6: Query Formulation Algorithm [69]

To implement the question classification, the researcher has realized the algorithm ideas (logistic regression with *Cross Validation*, *SVM Linear Kernel*) and test datasets (TREC data and manually generated dataset) of the research [70][71] in the implementation part. The common point of datasets have questions and labeled class of these questions. Questions are grouped with coarse-grained labels, which are *"Abbreviation"*, *"Entity"*, *"Description"*, *"Human"*, *"Location"*, and *"Numeric" (*Li&Roth [48]). Manually generated dataset that the researcher has 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 modeling. The question classification evaluated methods of the *Logistic Regression* and *Support Vector Machine* and the results can be found under Appendix A.13.

6 Experimental Results

In Section 6, the researcher will explain test methods and environments to create critical thinking about operator assistant web-based software. The researcher evaluates the OPC UA Client of the web-based software with some parameters, e.g., the viability of unit testing, mock testing, and load testing. The researcher separately evaluated the Semantic Question Answering with the precision of answers and usability of the question answering. The central motivation is to provide an assessment of performance for the web architecture. The OPC UA Client has complexities to test with mock testing because the establishment of a session at the protocol level for each request could cause failure on overall web-based software. A load testing might be used for testing functions of the web-based software, e.g., opening a session, sending a request, serialization of objects, and terminating a session. A timeout value of the testing tool and a timeout value of OPC UA Web Component should overlap; otherwise, results of performance or load testing can give the researcher wrong results in terms of failed requests. The Semantic QA needs various metrics as defined in Section 6.2. HTTP requests are equally crucial for performance tests and unit tests. Randomly selected requests with JWT authentication and their results are shown in Table 6.1. In the evaluation phase, chiefly, the researcher will ask three questions.

1) What is the accuracy of the question answering according to generated questions?

2) What is the performance and degree of functionality of the operator assistant web-based software?

3) How well does the web-based software react in case of balancing workloads?

6.1 Test Methods and Environment

Test environment covers the functionality, performance, and accuracy of the web-based software afore-mentioned questions in Section 6. Functionality and performance evaluation depends on the architecture behind the web-based software. A load test has been planned, and it was used to test web-based software against OPC UA performance to do so. Accuracy and relevant evaluation metrics will appraise the Semantic Question Answering. Manually generated test sets of the dynamic and static query have been used

to enlighten predefined metrics for the QA. OPC UA performance testing has still some delicacy in which they use mock data and unit testing with OPC UA Client. Therefore, the researcher decided to test with HTTP Request all the way through the front-end application and the back-end application to establish a deduction of hypothesis 1 (H-1). The load test was broken into two phases that each part has evaluated a different number of HTTP Requests.

At the first stage, the researcher has the following conditions to evaluate the performance of the web-based software with a single request without load balancing. This test will assess the core components of the web-based software. Then, the researcher changes the test environment by adding a load balancer. The load test takes a single request and multiple requests to test for each test case. In light of numerous requests, requests can be dispatched into one OPC UA Server, but each request is aiming to a distinct node in the server. The researcher realized the tests that have been formed in the *"Test Methods"* multiple times over again. For clarity, the researcher executes a single instance of the OPC UA Web Component and four instances with the load balancing.

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: The researcher sends a single HTTP Get Request to the Node OPC UA Server with a load balancer and without a load-balanced single core of the OPC UA Web Component 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, the researcher sends multiple HTTP Get Methods to the Node OPC UA Server with a load balancer and without load-balanced single core back-end application, (Kestrel embedded web server). Additionally, the researcher sends multiple requests to the Semantic QA with 10 users within 30 seconds to ensure the H-1.

Test Parameters: The researcher sets an HTTP header that uses the front-end application of the Angular 2+ as a proxy with HTTP Referrer. This referrer takes incoming requests as if it was coming from the front-end application.

Test Condition: All components including the front-end application and the Semantic Question Answering are up, and it sends a request like in Appendix B.13. Embedded firewalls and antiviruses remained offline.

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

For the functional tests, the researcher defined six HTTP requests that evaluate different functionality. These tests have been defined according to security and modularity. The researcher grouped the functional tests as insecure authentication in front-end, unappropriated HTTP method, and false token-based authentication.

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

The reason why the web-based software imposes mostly GET requests, which is about the idempotency and caching problems in HTTP requests. Furthermore, as being sent POST requests, the idempotency can affect the general results and POST requests that do not contain JSON serializations. Hence, the researcher may see more requests, but less of them are valuable for the purpose of evaluation. The researcher has never defined *"PUT"* and *"DELETE"* requests in the application; therefore, these requests are out of scope because of the data integrity and immutability.

6.2 Question Answering Data Sets and Evaluation Metrics

The Semantic QA has used well-founded evaluation metrics that other studies in the literature review. [Nguyen, Kosseim 2004] has evaluated their study *"Okapi Formulation"* often is called *"Okapi Weighting"* uses answer scoring. However, knowledge-based question answering systems do not regularly use an answer selection model because data sets are limited to obtain enough answers. Hence, the researcher will apply machine learning metrics which are called *"Precision"*, *"Recall"*, *"F1-Score"*, and *"Accuracy"*. The researcher has data generated by the *"Dynamic Server"* (Appendix B.5) which has size of 2.216 KB and 39374 lines. Other static data is originated by the *"eniLINK"* which has size of 19 KB and 73 lines.

[Diekema, Yilmazel, Liddy 2004] offered six 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 the researcher has 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(TP) stands for an answer to a question that is correctly labeled in case it is predicted positive [48]. The *False Positive(FP)* means that an answer that has incorrectly labeled false in case it is predicted true. *True Negative(TN)* defines that the answer is going to mark as a false classified value, but the actual situation is already false. Unlike the *True Negative*, *False Negative(FN)* states that an answer to a question would be false, but the researcher has predicted as true labeled.

$$Precision = \frac{TP}{(TP + FP)} \quad 1.9$$

$$Recall = \frac{TP}{(TP + FN)} \quad 2.0$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{(Precision + Recall)} \quad 2.1$$

$$Accuracy\ of\ the\ Model = \frac{(TP + TN)}{(TP + FP + FN + TN)} \quad 2.2$$

Manually generated questions are given in Appendix A.1, and a sample preview of a dataset as RDF/XML format is given in Appendix A.11.

6.3 Results

As illustrated in Table 6.1, the researcher has seven mock testings to test the functionality of the REST Interface. Results are evaluating the security and integrity of the message, for example, ID numbered with 1, 2, 4, and 5 evaluate the front-end application routing and error message according to the authentication mechanism. *ID 3* was tested multiple writing requests that are coming from front-end application with a referrer.

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 requests	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: HTTP Methods As per the Functionality

The researcher collaborates authentication schema, the semantic QA and the OPC UA Web Component of the web-based software to show results of HTTP Request that was undertaken by the practical implementation. Test *ID 6* responds with a *"Method Not Allowed"* message, which is a denial request that comes from the web-based software. The researcher has entirely six test cases of the functionality. *ID 1* test case is the initial authentication after taking username-password pairs. All other test cases must be contin-

gent upon the creation of a token that in case of authentication was granted. If the *Angular 2+* could not find the right token that the *OPC UA Web Component* generated, the front-end application should reroute to the login page. *ID 4* and *ID 5* test cases confirm the validity of JWT tokens in terms of the front-end application. *ID 7* test case is a proof for the restricted address space of OPC UA servers should send the *"Internal Error"* with a *"BadAccessRequest"*

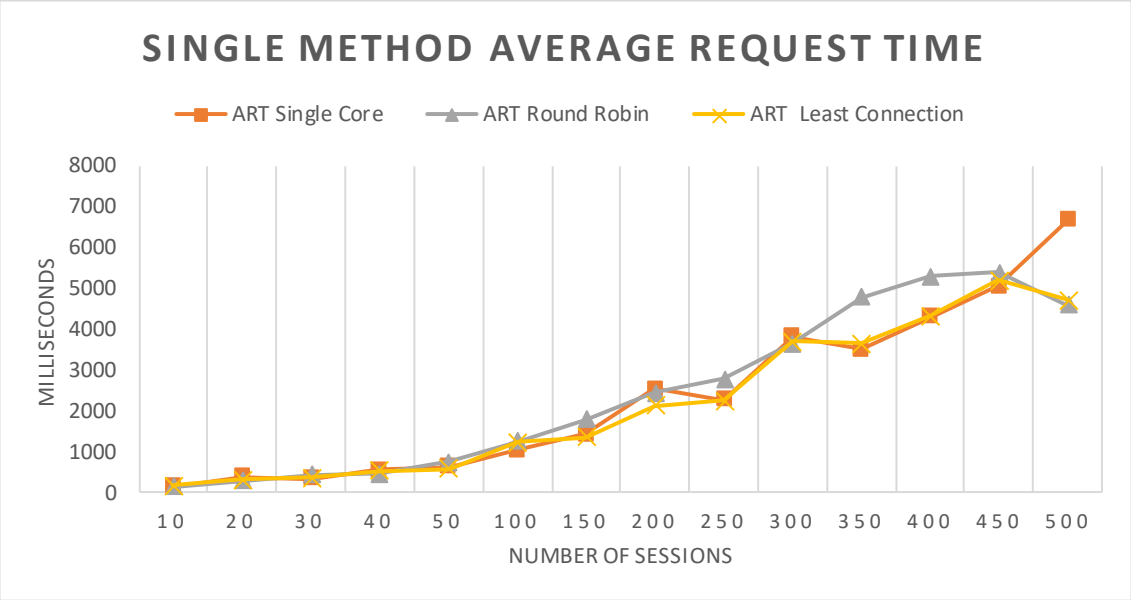

Figure 6.1: Single HTTP Request Response Time

As illustrated in Figure 6.1, the single core, the round-robin algorithm and the least connection inclined to rise below 7000 milliseconds. At the first 50 sessions, average request time does not change amidst the single core, the round-robin, and the least connection distinctly. As the round-robin was showing higher milliseconds until it reaches 250 sessions, single core and least connection were depicting similar inclination. Up to 250 sessions, load balancing algorithms closely tend to incline at the same rate. Interestingly, there is an overlapping point of 300 users for each method. The lower milliseconds of the average requests, the faster an end user of web-based software can obtain results. The single core of the back-end application substantially increased approximately 2500 milliseconds from 400 users to 500 users. Every 50 iteration of sessions, the round-robin scheduling makes a bumpy curve that starts from 350 sessions to 500 sessions. In the

end, there are 2500 milliseconds disparity on average request time between the single core and load balancing algorithms.

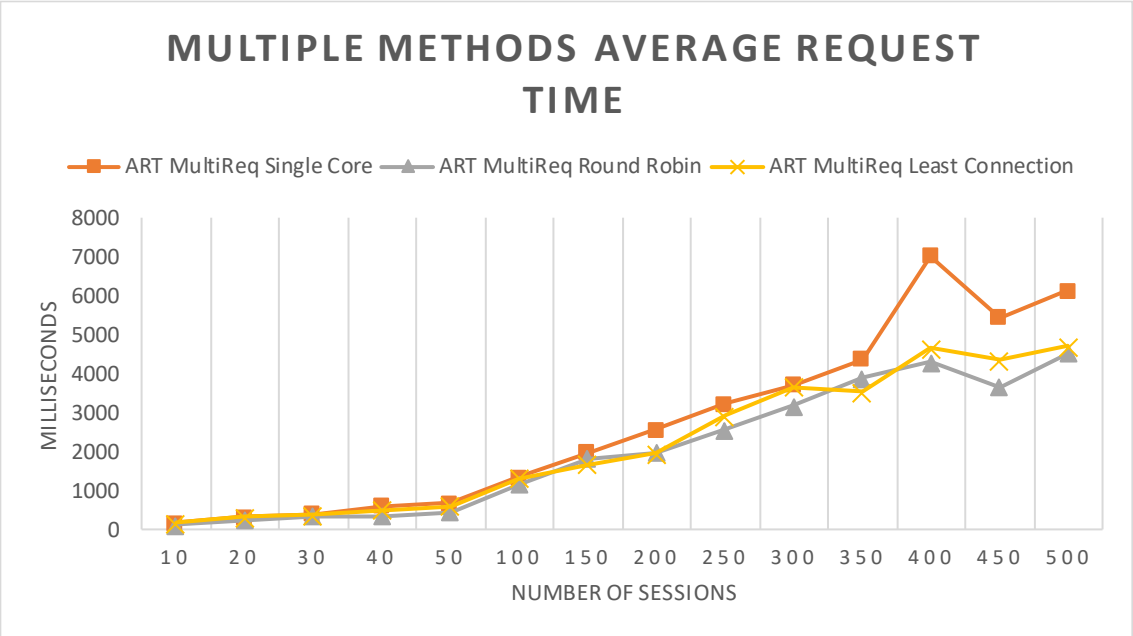

Figure 6.2: Multiple REST Request Response Time

In Figure 6.2, the *"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 core, which is that other algorithms of load balancing have never reacted similarly. The round-robin reached the lowest average request time. The round robin and least connection scheduling algorithms have operated within less duration that the single core does. While every 50 increments of the number of users affect a smooth step-up for the single core on a regular basis, the least connection made a zigzag movement starting from 200 users.

The round-robin displays lower average request time than the least connection and the single core does, which a query has been responded at most 4300 milliseconds under the round-robin scheduling algorithm.

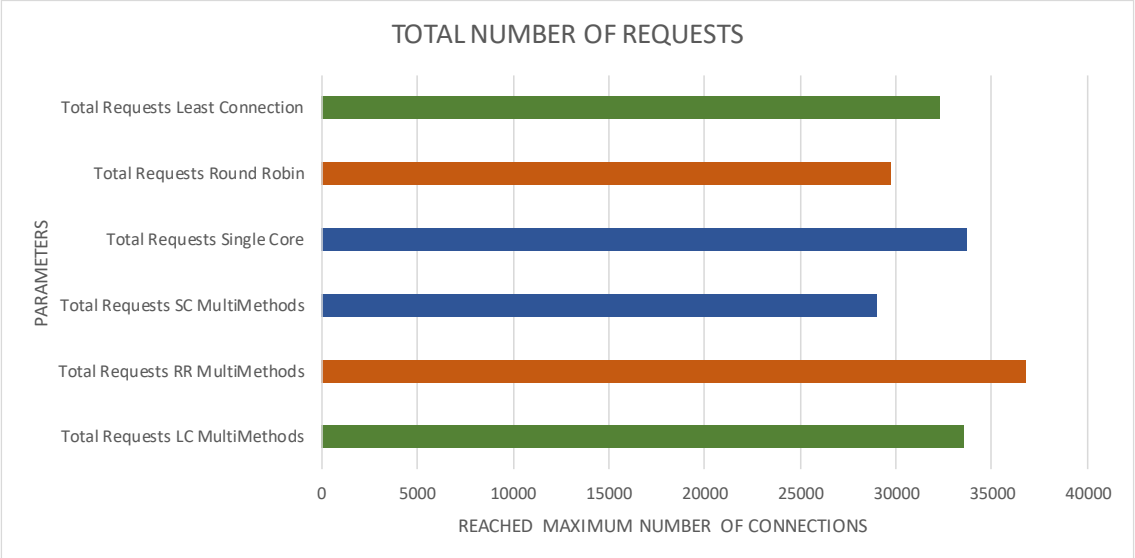

Figure 6.3: Total Amount of REST Requests

Upon modification of the architecture, the total amount of requests evaluation predicted the overall performance that concludes achievements in the sense of HTTP communication. In Figure 6.3, *"SC MultiMethods"*, *"RR MultiMethods"* and *"LC MultiMethods"* stand for *"Single Core Multiple Requests"*, *"Round Robin Multiple Requests"*, and *"Least Connection Multiple Requests"* respectively.

The round-robin scheduling increased 7000 requests, and the least-connection algorithm increased approximately 1254 requests under the load balancing. However, a total number of requests considerably dropped off around 5000 requests upon altering the architecture from non-load balancing to load balancing. Briefly, the round-robin increases %24.47 at total throughput; the least connection increases %3.73 at total throughput, the single core loses %13.87 at multiple HTTP requests decisively.

Accordingly, the single core reduced the number of requests significantly, when multiple requests were realized. However, the round-robin algorithm reached the highest amount of requests in the event when multiple HTTP methods have been sent.

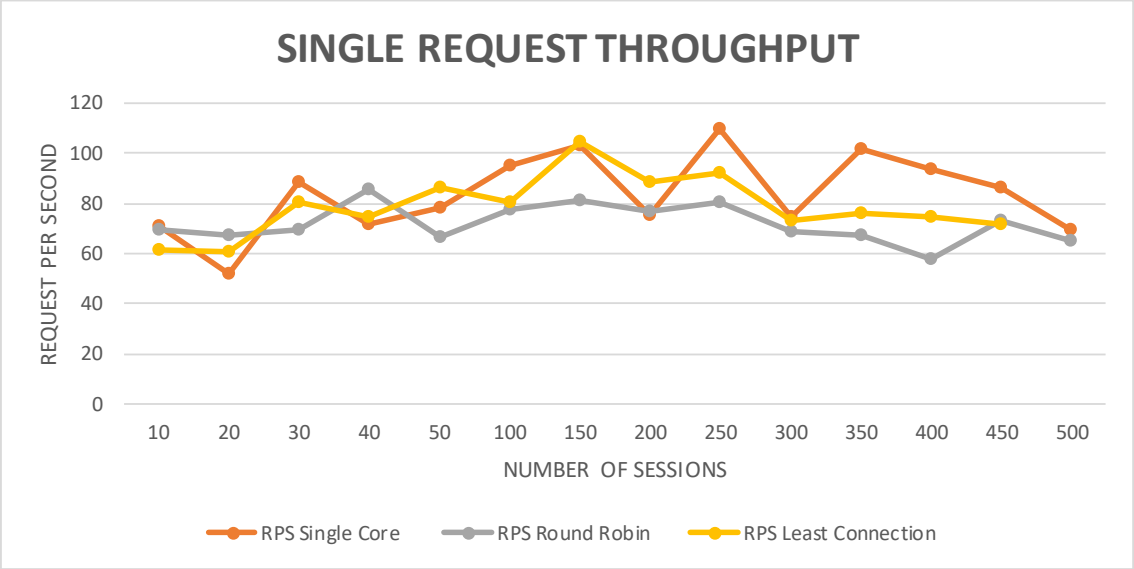

Figure 6.4: Single REST Throughput

As illustrated in Figure 6.4, the single core appears to be major fluctuations while creating request throughput. However, such fluctuations do not exist in load-balancing algorithms. After 350 users have been testing, the least-connection displayed lower throughput than the round-robin algorithm did. The fluctuations of the single core have been spotted three times above 100 requests per second. Neither the round-robin algorithm nor least connection could not achieve to create requests per second above than 100. In practice, the round robin and single core reach the same amount of throughput at 500 users, but the least connection reached 10 requests lower than the remainder of the methods made. Although the round-robin performed a steady path with a single HTTP request, the least connection and the round robin decreased the throughput per second after it passed 250 users. The single core reduces averagely 35 requests between 350 users and 500 users.

Consequently, the throughput per second is higher for the single core and the least connection algorithm than the round-robin algorithm. Although the round-robin algorithm generated the least amount of requests than other ways, the round-robin scheduling displayed a smooth line.

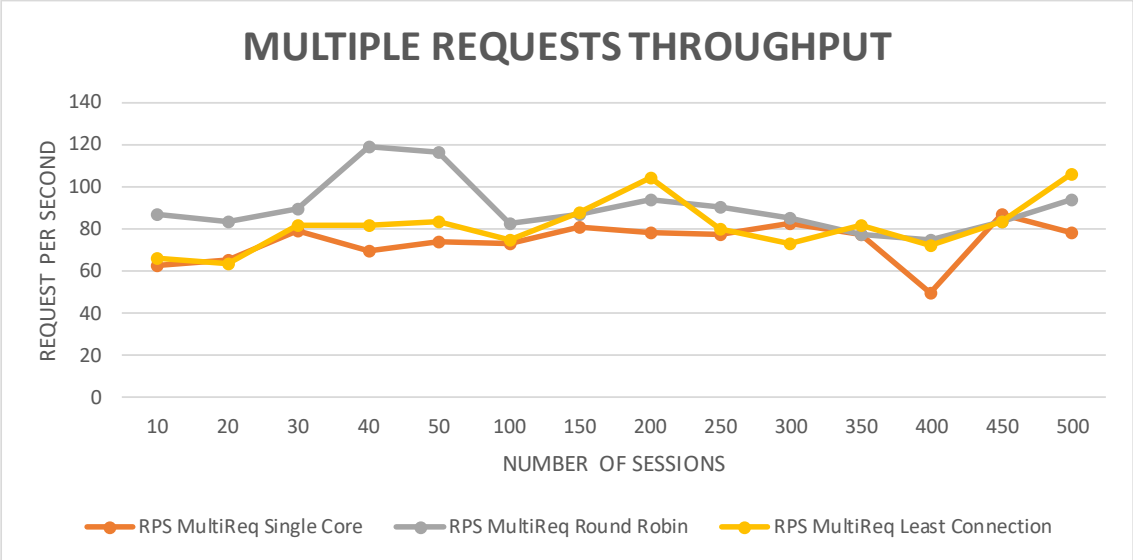

Figure 6.5: Multiple REST Throughput

As shown in Figure 6.5, the round-robin generates the multiple highest throughput at 40, 50, and 200 users. The single core generally keeps executing 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 software reached a peak of throughput approximately more than 45 requests as shown in Figure 6.4. The web-based software can respond to % 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.

Any tests have been evaluated the performance of OPC UA Component of the application took failed responses. This is because the researcher has equalized the timeout value of OPC UA Client (60000 milliseconds) to request timeout.

The following information concerns result for the semantic QA component of the web-based software. The Semantic QA has a performance result called *"Answer Return Rate,* which is a similar test parameter as the researcher realized a performance test for the OPC UA Web Component. Consecutive query and the initial query for dynamic and static queries have disparities aspect of diversified round-trip time. Theoretically, there is no remarkable difference between static query application and generated data appli-

cation. In practice, having bootstrapped loading the word vectors and universal dependencies, the practical implementation could avoid repetitive loading for each query in order to eliminate round-trip time differences. For that, reasons that were mentioned before, the researcher preferred to give average answer-return rate results for dynamic and static queries.

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 the *eniLINK* – 18.92 seconds average QA against dynamic query from streaming data – 17.48 seconds QA against Open-Domain QA [72] – 20.55 seconds
Querying Style	Keyword-Based Search and Semantic Search
Coverage	The *eniLINK* data, the *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 web-based software does not cover the open domain QA, but the researcher compared with a result of answer return rate of a pattern-based open domain QA. In practice, this type of question answering should have been faster than our approach in Section 5.3.5. It shows similar round-trip time as much as the semantic question answering does. Data size is relatively changeable between a static search and dynamic search. The dynamic search exploits streaming data generated by time series data from a key-value database.

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

allow 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. The Query Formulation Assistance is a result of ergonomic and cognitive compatibility of the semantic QA.

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

The researcher has obtained the *F1 Score*, *Precision*, *Recall,* and *Accuracy of the* Model (see 1.9, 2.0, 2.1, and 2.2). The researcher classified responses of the questions as four labeled, which is named multi-class classification. The researcher does not have any true negative values; hence, the researcher did not put the true negative column into the table. 34 questions have been correctly answered out of 50 questions. As can be seen, it is obtained a high precision value, but the *Recall* value is more important for evaluation. Under these circumstances, he will lose the right answers of %72.34 questions. If the researcher has 100 questions that should be answered, the QA will miss the correct answer at most %27.56 of the retrieved answers. The precision value %94.44 indicates that the researcher has built a classifier with high precision. *F1-Score* is being perceived as a balance factor between the precision and the recall. Since the precision and recall values are high, the semantic QA answering may be less trade-off among the recall and precision.

Consequently, manually generated data has shown the overall performance against *eni-LINK* streaming data and created data as above in Table 6.3.

7 Discussion

In the beginning, research questions and hypotheses were defined in order to elucidate the research goals. The researcher will summarise the key findings according to the results of the experimental results. Since the synthesized idea that has been presented as transdisciplinary research, there was no particular study for a human operator assistant solution combining with web science, artificial intelligence, and industrial networking. However, the researcher will argue for the results carefully that relate to the state-of-the-art studies that the researcher has disputed in Section 2. Conclusively, the researcher will elaborate on the implication of practical results about how this affects our original hypotheses in Section 7.3, and then the researcher concludes the research in Section 8.

7.1 Summarization of the Key Findings

The first finding was relevant to the OPC UA web component; it has been standardized on communication protocols and service sets. Nevertheless, there is no common architectural implementation aspect between clients and servers. For instance, some OPC UA Servers have strict protections regarding session establishment and size of the view of address space (see RQ-1).

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. A developer can decide this trade-off that is according to requirements in a division of smart factory (see RQ-2).

The third finding was that the performance of the human operator assistant software could be enhanced with balancing workloads. Every manufacturing division has a different necessity as per the number of users and their functional expectations. The total amount of requests and throughput per second would be higher with the round-robin and the least-connection load balancing algorithms against multiple HTTP requests. The algorithm of distributed computing selection is equally important as much as being implemented a load balancer (see RQ-4).

The fourth finding was that stateless communication and stateful communication are still hard to integrate into the web-based software. Even a wrongly set timeout value of the OPC UA Web Component can increase the network failure, which leads the web-

based software to the network bottleneck. Moreover, establishing a distinct session for each request can cause the overflow to an OPC UA Server if there is no protection against session workload overflow. One can solve the problem of sessions via session-less invocation of the service sets. Thanks to a session of OPC UA, it ensures the namespace index within a particular session lifetime, this kind of solution would have a trade-off between consistency and performance (see RQ-2, RQ-4, and H-2).

The fifth finding was that the semantic QA depends on the type of domain and data set. If the domain and data sets were changed in a different division of a smart factory, a property semantic extraction algorithm would be a necessity to map these properties onto *"<subject, predicate, object>"* triples (see RQ-3 and H-2).

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 model-based machine learning methods. Even if one may implement these solutions above, high computational power is a necessity to find out complex morphological elements of a semantic query and to extract a model from complex data sets (see RQ-2 and H-2).

The seventh finding was related to the discovery service no matter a local discovery server or a global discovery server; the human operator assistant software needs a discovery server to find the endpoints of OPC UA Servers and to append certificates into fixed sessions. Remote domains that attached to different manufacturing environment or different facility should have an organization point as service discovery (see RQ-4 and RQ-1).

The eighth finding was that establishing a session has a big overhead for the web-based software while realizing concurrent requests. To solve this problem, *"secureChannel ID"* can be tracked by a sticky session of a load balancer. A sticky session can follow by assigning tracking ID to a peculiar session. Another solution could be observable multi-thread session pool that has been designed with observer pattern. In addition, synchronous HTTP calls are reducing the number of requests without balancing the workloads and increasing average request time of the requests. This problem can solve with asynchronous HTTP calls, a thread pool with a load balancer, or a session-thread pool at the initial connection (see H-1 and RQ-2).

The ninth finding was that established domain weighting is necessary for the semantic QA. The researcher has separated into three different domains as *"dynamic query"*, *"static*

query", and *"OPC UA related query"* respectively. To overcome this issue, a domain-ranking module can be added through the master domain and slave domain by calculating the term frequency of the questions (see H-3 and H-2).

During the load tests, the web-based software displayed a couple of unexpected results. Firstly, the web-based software created fluctuations as illustrated in Figure 6.4. These fluctuations start from the beginning of user request until 300 users have sent requests. Unexpectedly, the web-based software did not show any fluctuation behavior under multiple HTTP requests in Figure 6.5. Thinking as a whole, Figure 6.5 should have displayed a 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 (see RQ-1, RQ-2 and H-1).

The second unexpected result has occurred in Figure 6.3. Usually, a higher number of HTTP requests should generate an HTTP response in a particular duration of time. In Figure 6.3, the single instance that has been worked by OPC UA Web Component returned fewer results after the system applying multiple HTTP requests. A potential outcome is that the OPC UA Web Component is more successful with load balancers to prorate workloads than the software does without load balancing while applying numerous HTTP requests (see RQ-1, RQ-2, and H-1).

The third unexpected result was the semantic QA could reduce the number of total requests of the OPC UA Web Component regarding multiple threaded environments under a heavy load testing. Whether a typical load balancer is used to compute the load of requests for distinct web components, the risk of a single point of failure might be increased. The problem is here that the semantic QA works synchronously, unlike the OPC UA Web Component. The entire application runs stable, even though any error case has not been observed, but the web-application generates a relatively small amount of the total number of requests according to the experimental results. In addition to the third unexpected result, if the messages of HTTP response are sent without serializing JSON format, the semantic QA may throw a socket connection error because of the *carriage return character* at the end of the messages. The solution of this problem is to provide good abstraction of HTTP requests with JSON serialization (see H-3, RQ-4, and H-2).

The web-based software can remedy the first and second unexpected results by using a load balancing with worker processes. For instance, a load balancer spawns multiple processes with their working threads. When multiple HTTP requests arrived at the web-based software, the load balancer put a chunk of requests into a particular process and

send it to the back-end application. However, insertion of a thread into queue should have another stage like dequeuing from the queue of a thread pool. Repeated test cases could make an inequality on thread creation while thread lifecycle changes from idle to working in the thread queue. Such unexpected behaviors can reduce the maximum reached connections and create major fluctuations on HTTP requests.

7.2 Challenges in the Practical Implementation

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

The second challenge was related to the representation of the multiple data from multiple monitoring nodes in a stateless protocol is hard. Instead of using two-way data binding of *"Angular 2+"*, the front-end module of the web-based software can connect to the aggregated server that holds a monitoring node in a different hierarchy.

The third challenge was that the heuristic based syntactic parsing could not produce a model just as a supervised machine learning method does. It should start from the beginning as shown in Figure 5.3 and all process takes a considerable amount of time, which shows a lower degree under a performance test. In order to solve this problem, template-based questions – SPARQL query pairs can be realized. However, this solution reduces the expandability and ease of use. Other solutions 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 algorithms.

The fourth challenge was the creation of manually generated questions and data sets from the Dynamic Server quasi OPC UA Server. Due to the difficulty of natural language processing, manually created questions cannot 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. Furthermore, increasing the sampling size from the same data source is hard. Such sampling size impeded 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 a clear boundary of each predefined class. Another solution is to examine not only with accuracy, but also with the 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 of as a clear test case since the nature of application complied with the microservice architecture. Another issue about the testing of the 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 activities, 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 the OPC UA Services. At the protocol level, [Grüner, Pfrommer & Palm 2015, 2016] stated the use of stateless transport layer protocol instead of the OPC UA could reduce communication effort between clients and servers in case of eliminating session establishment phase.

RQ-2 refers to an architectural design thinking for modularity and scalability. Having inferred results from the Experimental Results, the case of multiple components should serve to a common goal, and the architecture might be divided into the smallest possible parts. For instance, if there would be a dependency between the Semantic QA and the OPC UA Web Component, the web-based software can face potential threats towards robustness, security, ease of use, and performance.

RQ-3 and H-2 have a couple of results that are closely related to each other with regards to the linked data source, and manually generated set of questions. It is implied that the manually created set of questions could have a bias depend on users who these questions set made. External data and feature engineering by comparing and calculating the most common dataset in the research circle can alleviate the problem when the results of the Semantic QA in Table 6.3, the Precision, Recall, F1-Score, and Accuracy appears to be propagated in a correlation. Accurate results have strongly connected the parameters of

evaluation as mentioned earlier; conversely rapid results from the Semantic QA is related to Table 6.2. It is shown in that table; all kind of answer return rate has been finished below 25 seconds.

RQ-4 refers to generalization, expandability and scaling issues regarding overall architecture. Due to the OPC UA platform-independent design, the OPC UA Web Component can easily be generalized in a different manufacturing facility or division. However, the Semantic QA has a couple of dependencies that come from the data source of generated data, streaming data source key-value mapping and ontology predicate mapper. Ontologies have various type of predicates which can represent a noun phrase and verb phrase — making this process automatic, which the process can be possible with data standardization from generated and streaming source. However, several semantic QA systems may be developed for different domains, and these QAs can connect a core question answering for operations, such as the classification, the parsing, and the normalization process.

The Semantic Question Answering and the OPC UA Web Component can clarify separately to H-3. The Semantic QA may port a framework from other domains, but this causes increasing code complexity and reducing modularity of the semantic QA as offered by [Molla, Vicedo 2007]. [Molla, Gonzalez Et al. 2007] stated that a restricted domain question answering might be complicated and practical.

Theoretically, a question answering can employ each possible algorithm that has mentioned in Section 2.2. In practice, the semantic question answering compels constraints such as the answer return rate, data size, data coverage, and question limitation. The Semantic QA can give rigorous and correct results offered by [Diefenbach Et al. 2017], [Dwivedi, Singh 2013], [Palaniappan, Sridevi & Subburaj 2018], and [Tatu Et al. 2016] by H-2. The lemmatization, named-entity recognition, syntactic parsing can identify the contiguous span of phrases. [Ferre 2012] controlled natural language approach is hard to construct but it would reduce ambiguities of natural expressions by adding a quasi-pattern into the questions. [Celikyilmaz 2006] and [Giannone, Bellomaria & Basili 2013] proclaimed that the HMM unsupervised and other supervised machine learning algorithms require to broaden data coverage, high data size, a considerable amount of time to train and avoiding overfitting and underfitting issue. [Luz, Finger 2018] highlighted that the recurrent neural network and artificial neural networks might not need to deal with linguistic knowledge or development of complicated grammar, but the systems require tests with different datasets to guarantee to create matched syntax for query formulation.

As a consequence, every algorithm has a different nature and domain, which algorithms demand appropriate natural language processing method to create a question answering. The results of applications that the researcher overviewed are slightly comparable to this research.

When the point comes to the OPC UA Web Component, [Paronen 2015] entailed that the front-end application can do pre-emptive cache frequently request node to increase overall performance. This thesis does not agree with this approach because the caching can save a response without updating the data and then it may show the outdated response to a user. Caching the credentials may cause a security leak, where the web-based software generated can be displayed to an unauthorized user. Furthermore, extra-configured proxy processing of a caching mechanism may increase the average request time. As to solution of the problem in mapping a stateful OPC UA session onto a stateless session, the thesis would critique about assigning workloads to the same session ID by a load balancer.

[Cavalieri, Salafia & Scroppo 2018] concluded REST integration with read & write and subscription may be enough to compose a web application. However, without architectural design thinking and front-end applications development, developing a web application for OPC UA is a complex and challenging process. They have designed detailed *Subscription* and *Monitoring Node* concepts, but *aggregated OPC UA Server* that provides persistent session storage between clients and servers are more suitable for full-fledged application development. Besides, they did not offer a solution regarding mapped session-less protocol onto session-based communication. [Grüner, Pfrommer & Palm 2015], [Grüner, Pfrommer & Palm 2016], and [Shiekofer, Scholz & Weyrich 2018] have mentioned more about the comparison between the transport layer communication protocols and REST service integration.

Lastly, the proposed approach is quite useful for the implementation and design of the web-based software in the state-of-the-art section. The thesis has investigated the methods as the researcher discussed above in the relationship of the *Experimental Results*. The aspect of web science of architectural design, this thesis did not examine previous researches due to the lack of matching study related to the primary research goal.

8 Conclusion

8.1 Summary & Future Works

This thesis has undergone the realization of a web-based solution for smart factories that created new adoption of web science, artificial intelligence, and industrial communication. It was accomplished by synthesizing idea relevant to transdisciplinary researches. These achieved has been overviewed with theory and practice regarding the *"Semantic QA"*, the *"OPC UA Web Component"*, and the *"Heterogeneous Linked Data Storage"*.

The initial stage of the research was to introduce that the problems which human operators have faced in the context of manufacturing, operation and training experts and operators in a smart factory. In Section 2, the thesis examined the problem by separating three main titles that are tightly connected. In Section 3 and Section 4, the theoretical background of human operators, the OPC UA, its Service Sets, and the semantic QA have been overviewed. Having observation on the service sets of OPC UA, multiple requirements that have been utilized in the thesis were acquired. While the discovery and subscription services were dissected the aspect of architectural design thinking, the address space and information model have given an overview in the context of linked data and core functionality of the web-based software. Besides, the thesis examined valuable linked data formats, linked data preparation for streamed time series data and the *"Information Model"* referenced-data. Neatly, the core functions of the semantic question answering have been investigated with natural language methods. This examination has been considered to syntactic and semantic similarity, probabilistic syntactic parsing, question classification, query formulation, and normalization stages. With the flowing of steps, the researcher provided the details of the theoretical background that the researcher has been using the following section.

Section 5 dealt with the implementation of the web-based software in considering with front-end development, web architecture development, back-end development, and the semantic QA development. By implementing the heuristic based syntactic parsing algorithm, a human operator can get a concise answer for factoid questions from the web-based software. Section 6 shows results to prove about research questions and hypotheses. These experimental results demonstrated the achievement of high precision and high recall classifiers, that are, depending on manually generated questions in the restricted domain question answering. Extendibility of the application depends on the

usage of multi-purpose services, such as discovery; load balancing, and OPC UA direct data access service sets. Principally, several limitations could be listed, which they could alter the scope of the thesis. The lack of question test datasets and semantic datasets restrains the overall results. Lack of standardization of OPC UA Servers can change the implementation of service sets in the web-based software. Lack of data coverage, size, and annotation prohibit the use of deep learning methods or supervised methods that create a classifier to match linked data triples to questions. The limited coverage of the data caused a low amount of sampling size; hence this affects the result of the semantic QA in terms of *imbalanced dataset* problem. Next, the researcher cannot aim at applying all of the service sets that the OPC UA realized. Thus, the researcher has limited the RESTful API with discovery service, read, and write requests, and subscription with a monitored node. A limited number of researches for the OPC UA web component and the OPC UA data generation was also another drawback.

For future improvements, firstly, the proposed semantic QA should do the natural language processing step for time-constrained tasks. Secondly, the proposed synchronous semantic question answering should employ tasks with asynchronous communication in a non-blocking queue. A non-blocking input-output queue can help to alleviate the problem of synchronous calls. Thirdly, automated feature module can identify predicates, object, and subject through the standardized data model in order to create a proper dataset from OPC UA Information Model. Fourthly, an aggregated server can be planned the future implementation onto the *Dynamic Server*. Hence, the researcher can display monitored nodes with RESTful web-based software responsively, and the web-based software can connect a single server to send queries to multiple root nodes. Fifthly, the quality of predicates that the *eniLINK* defined can be improved. Notably, the researcher can add more property descriptions and corresponding explanations through a web application. Sixthly, authorization can ensure a role-based authentication after authenticating in the web-based software. These roles can be broken into administrative and user roles. A manager component can assign different rights to these roles to provide enhanced security. Seventhly, a question autocomplete system can be designed. Such a system would be efficient because it prevents the obligation of pattern or template based question types. By scoring correctness of answers, the system can give an excellent insight to users what to type. Lastly, a named entity recognition for smart factory lexicons can be added in order to eliminate a set of natural language processing steps. The more qualified data has been inserted in the smart factory, the higher accurate reason induction that is compatible with the QA can be realized.

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*, 2019. [Online]. Available: https://opcfoundation.org/about/opc-technologies/opc-ua/. [Accessed: 16-Nov-2018].

[6] S. Cavalieri, M. G. Salafia, and M. S. Scroppo, '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. Scroppo, '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. Tatu, T. Erekhinskaya, 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. Giannone, V. Bellomaria, R. Basili, and I. Department of Enterprise Engineering, University of Rome Tor Vergata, Roma, 'A HMM-based Approach to Question Answering against Linked Data', *Comput. Sci.*

[23] C. Unger, L. Bühmann, 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.

[24] 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.

[25] 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 LNAI, pp. 11–25, 2012.

[26] A. R. Diekerma and E. D. Liddy, 'Evaluation of restricted domain Question-Answering systems', *Cent. Nat. Lang. Process.*, pp. 12–16, 2004.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] D. Anicic and P. Fodor, '[epsparql] EP-SPARQL: a unified language for event processing and stream reasoning', *Proc. 20th …*, pp. 635–644, 2011.

[34] D. F. Barbieri, 'C-SPARQL : SPARQL for Continuous Querying', *Proc. 18th Int. Conf. WWW*, vol. 427, no. c, pp. 1061–1062, 2009.

[35] H. Hasemann and A. Kröller, 'The Wiselib TupleStore: A Modular RDF Database for the Internet of Things', *J. Phys. Soc. Japan*, Apr. 2018.

[36] Universita degli Studi di Catania, 'OPC UA Web Platform', 2017. [Online]. Available: https://github.com/OPCUAUniCT/OPCUAWebPlatformUniCT. [Accessed: 14-Jan-2019].

[37] 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.

[38] R. Margaret and D. Daniel, 'Definition of Smart Factory'. [Online]. Available: https://searcherp.techtarget.com/definition/smart-factory. [Accessed: 05-Dec-2018].

[39] 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].

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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].

[46] Mariusz Postol PhD. Eng. (Project Manager), 'OPC UA Information Model Deployment', Wolczanska, Poland, 2016.

[47] 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.

[48] 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.

[49] P. D. Leslie F. Sikos, *Mastering Structured Data on the Semantic Web*. Berkeley: Apress, Berkeley, CA, 2015.

[50] 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.

[51] Pure Python OPC-UA Client and Server, 'Free OPC-UA Library'. [Online]. Available: https://github.com/FreeOpcUa/python-opcua. [Accessed: 22-Nov-2018].

[52] TU Dresden, 'Plt-TUD'. [Online]. Available: https://github.com/plt-tud/opc_ua_xml_export_client. [Accessed: 22-Nov-2018].

[53] F. Breitling, 'A standard transformation from XML to RDF via XSLT', *Astron. Nachrichten*, vol. 330, no. 7, pp. 755–760, 2009.

[54] C. D. Manning, 'Foundations of Statistical Natural Language Processing - Christopher D. Manning', pp. 1–704, 2005.

[55] A. Clark, C. Fox, and S. Lappin, *Computational Linguistics and Natural Language Processing*. West Sussex, United Kingdom: Wiley-Blackwell, 2010.

[56] A. Taylor, M. Marcus, and B. Santorini, 'The Penn Treebank: An Overview'.

[57] J. Perkins, D. Chopra, and N. Hardeniya, *Natural Language Processing : Python and NLTK*. 2016.

[58] neo4j, 'The Jaccard Similarity Algorithm'. [Online]. Available: https://neo4j.com/docs/graph-algorithms/current/algorithms/similarity-jaccard/. [Accessed: 16-Jan-2019].

[59] 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.

[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] 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].

[62] J. H. Skeie, *Ember.js in Action*. New York: Manning Publications Co., 2014.

[63] M. Tielens Thomas, *React in Action*. Manning Publications Co., 2018.

[64] S. Hochhaus and M. Shoebel, *Meteor In Action*. New York: Manning Publications Co., 2016.

[65] A. Lock, *ASP.NET Core In Action*. New York: Manning Publications Co., 2018.

[66] TechEmpower, 'TechEmpower Framework Benchmarks'. [Online]. Available: https://github.com/TechEmpower/FrameworkBenchmarks. [Accessed: 23-Jan-2019].

[67] OPC Foundation, 'OPC Unified Architecture .NET Standard'. [Online]. Available: https://github.com/OPCFoundation/UA-.NETStandard. [Accessed: 13-Mar-2019].

[68] W3C, 'Xpath Tutorial'. [Online]. Available: https://www.w3schools.com/xml/xpath_intro.asp. [Accessed: 14-Jan-2019].

[69] O. Oruc, 'Question Answering Source Code', 2019. [Online]. Available: https://github.com/zointblackbriar/QuestionAnswering. [Accessed: 07-Apr-2019].

[70] S. Khare, 'Question Classification Implementation'. [Online]. Available: https://github.com/swapkh91/Question-Classification. [Accessed: 26-Feb-2019].

[71] 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].

[72] E. Andrawos, G. G. Berrotarán, R. Carrascosa, L. A. i Alemany, and H. Durán, 'Quepy - Open Domain Question Answering'. [Online]. Available: http://quepy.machinalis.com/. [Accessed: 15-Apr-2019].

[73] 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.

[74] C. Richardson, *Microservices Patterns with examples in Java*, First Edit. New York: Manning Publications Co.

[75] Arnon Rotem-Gal-Oz, *SOA Patterns*, First Edit. New York: Manning, 2012.

[76] OPC Foundation, 'OPC Unified Architecture Specification Part 14: PubSub Release', 2018.

[77] E. Rossignon, 'Build OPC UA Applications in JavaScript and NodeJS', 2019. [Online]. Available: http://node-opcua.github.io/. [Accessed: 10-May-2019].

Appendix A Semantic QA

A.1 Manually Developed Test Questions – Precision and Recall

Question	ID	Precision	Recall
What do linkedfactory, heatmeter, and e3fabrik incorporate exactly?	1	0.0	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 of this company. Could you tell me please what the value of sensor1 of machine1 is	5	0.0	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	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	0.0	0.0
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	1.0	1.0
What is the value of sensor1 in machine6?	21	1.0	1.0
What is the minimum that we can calculate for sensor1 of machine1?	22	1.0	1.0
What is the value of the 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	1.0	1.0
Could you get me the references of nodes?	27	1.0	1.0

Could you browse generated data?	28	1.0	1.0
Is the E3-Sim member of linkedfactory?	29	0.0	0.0
Could you take me all members of generated data?	30	0.0	0.0
Give me all registered node id	31	1.0	1.0
I need to learn parent node id in generated data	32	0.5	0.5
Could you give me parent nodeID in the file of generated data?	33	1.0	1.0
Give me all data blocks	34	1.0	1.0
Data blocks in generated OPC file	35	0.0	0.0
Give me the name of stations in generated data	36	0.0	0.0
All stations which are in generated data or new data	37	0.0	0.0
Registered node id	38	0.0	0.0
Who is Fofab?	39	0.0	0.0
How is the system status for sensor1 in machine1?	40	1.0	1.0

Table 0.1: Precision and Recall of Answers

A.2 Natural Language Understanding Libraries

TextBlob: TextBlob is a tool based NLTK to process natural language expressions without occurring NLTK's function overhead. It was written in Python 2.x, but it is also com-

patible with the Python 3.x versions. It provides a simple application programming interface for the purpose of utilizing common tasks of natural language processing such as part-of-speech tagging, sentiment analysis, and classification.[11]

Stanford CoreNLP: Stanford University presented one of the fastest and robust libraries for natural language processing. The only drawback of this library has limited support for *Python* programming language. However, it has been solved this problem with the RESTful – compatible web service by sending external HTTP queries from Python programming language.

The Stanford CoreNLP works based on the *Java Virtual Machine* so that it can be conceptualized as model-view-controller pattern. Stanford CoreNLP supports a diversity of implementation that is a vital role for natural language processing. It provides a set of functions and procedures that allow performing tasks which are annotation-based resources that are suitable with different languages [73]. The main drawback of the CoreNLP is that when a developer needs to use different programming language than Java, the developer should wrap up the Java compiled packages to specific other programming languages. This wrapping-up process reduces support of full-features such as sentiment analysis, dcoref, regexner [12].

SpaCy: It is an open source library for natural language processing which was written in Python and C-counterpart Cython[13]. 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*, the library 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 that is compatible with Python programming language. The AllenNLP also provides a demo tool which has used in this work to demonstrate the development steps of NLP. The AllenNLP has advanced features to use, which is not only industrial scale applications but it also has scientific purpose tools such as coreference resolution, semantic role labeling, open information extraction or textual entailment.

[11] https://textblob.readthedocs.io/en/dev/
[12] https://github.com/Lynten/stanford-corenlp
[13] https://spacy.io/api/

SyntaxNet: It is a library provided by Google that works with a deep neural network based on the *"Tensorflow"*. The purpose of this library is to catch the highest predicted accuracy for a syntactic parser and dependency parsing. Moreover, this library focuses on dependency parsing more than parsing phrases and clauses which are done by the constituency parser.

Natural Language Toolkit: The *"NLTK"* is one of the basic language toolkits 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	No support for interoperability (Only in Windows OS)
Stanford CoreNLP	Strong support for a variety of languages	The central point of failure, Bottleneck if there is not enough maintenance
Spacy	Easy to use with web platform technologies Faster syntactic and dependency parsing	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 interoperability (Only in Windows OS)
SyntaxNet	Entirely 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

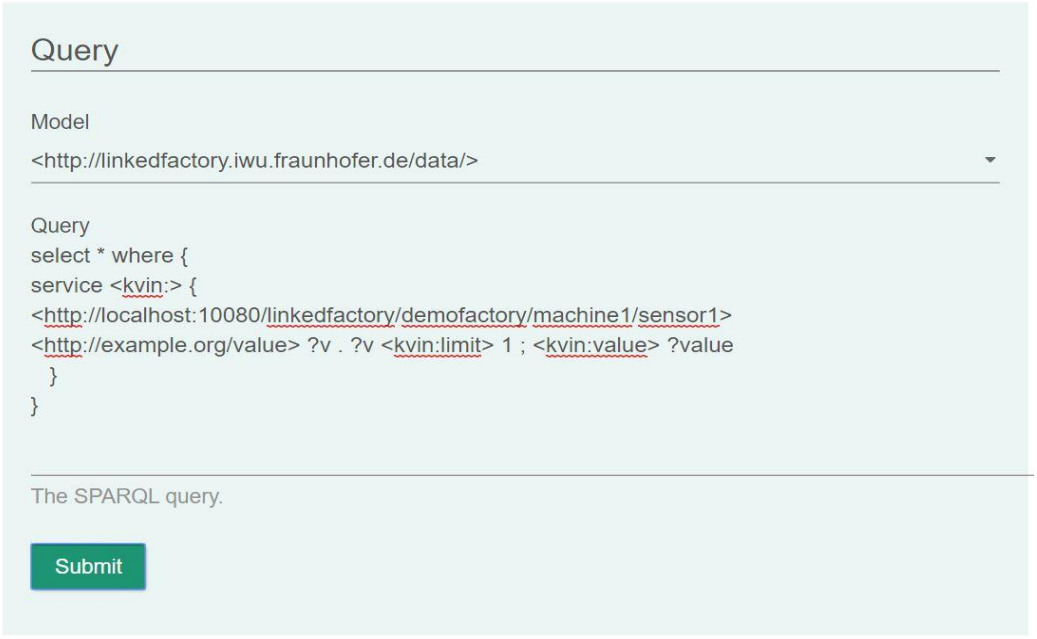

Figure 0.1: eniLINK Sample SPARQL Query

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

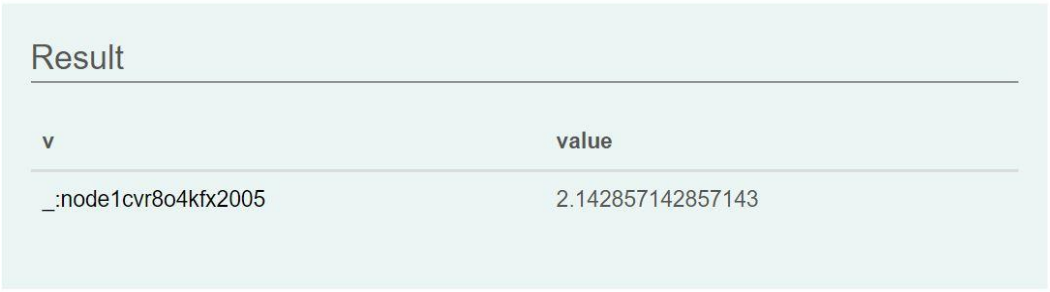

Figure 0.2: A Result of Continuous Data

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

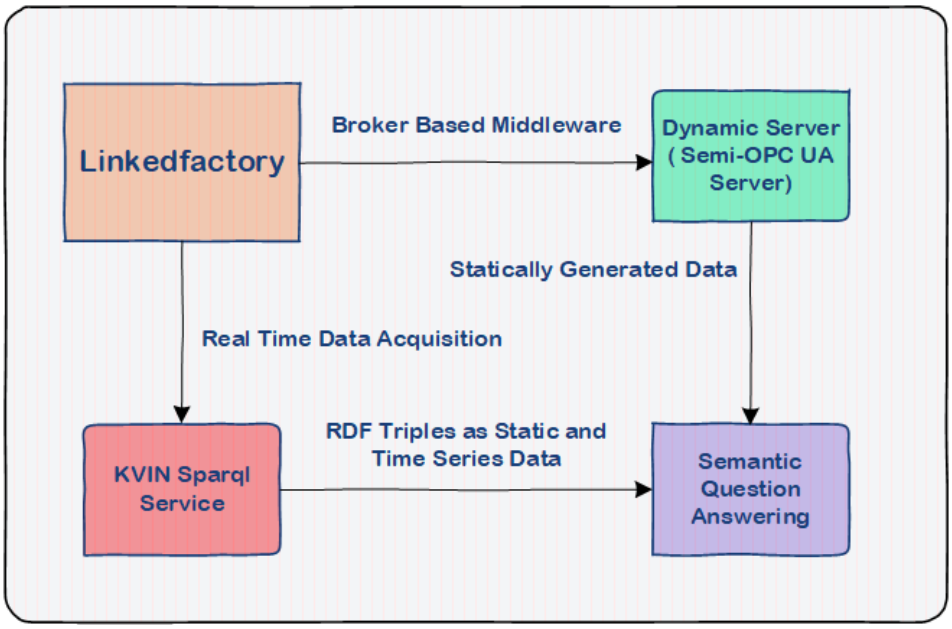

Figure 0.3: KVIN Service Relationships with Components

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 Sample SPARQL against eniLINK Properties

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

Listing 0.3: Sample SPARQL against Generated Local Triples

A.9 Deep Parsing-Shallow Parsing

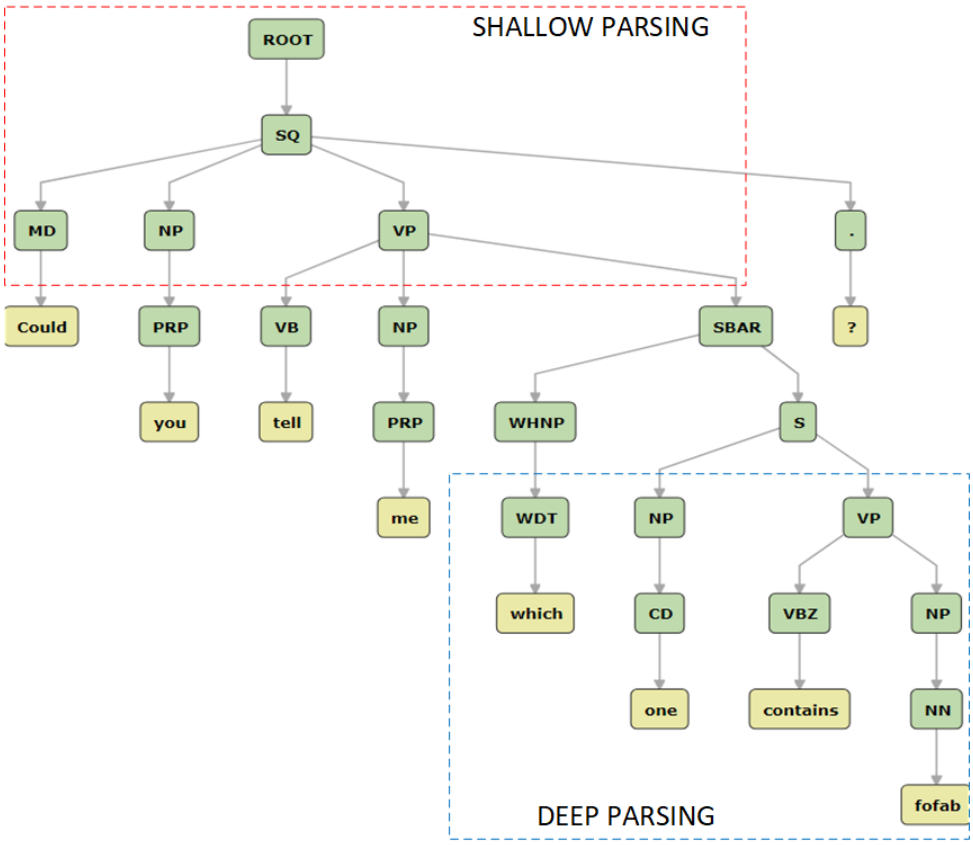

Figure 0.4: Constituent Parse Tree for Factoid Questions

A.10 An Answer Against the OPC UA Information Model

```
(rdflib.term.Literal(u'linkedfactory.iwu.fraunhofer.de/linkedfactor
y/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/linkedfactor
y/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 0.4: An Answer from Generated OPC UA Semantic Data

A.11 Transformed Turtle RDF Data

```
@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 0.5: Preview of Generated Semantic Data from an OPC UA Server

A.12 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 0.6: Transformed XML Data into RDF/XML

A.13 Question Classification

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 Regression 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: The logistic regression is a predictive analysis method that uses a binary classification method wrapped the combination with range [0, 1]. The method of *newton-cg* employs L2 penalization and multinomial distributions. To classify more than one categories, multinomial logistic regression method. Regression models are useful for continuous data; however, they can be used when required a categorical dependent variable.

Logistic Regression with lbfgs: *Limited Memory with BFGS* is an optimization algorithm of Newton-methods. The algorithm attempts to minimize the error with the gradient of the objective function. **Logistic Regression with Cross-Validation** defines the same dataset as training and test data. All of the methods that have mentioned have a common purpose, which is finding a global minimum to reduce train error.

Linear Support Vector Classification: The 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. For the sake of speed, the practical implementation employs the linear kernel. It should be remembered; although the linear kernel is faster than Gaussian kernel, the Gaussian kernel gives more predictive results.

Appendix B Web-Based Software

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

Scalability, ease of use, and modularity are substantive factors of a web architecture for a web-based software application in smart factories. Generally, applications that have targeted particular tasks have been tended to develop 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, web-based software can be a lifesaver to make the mobility of tasks possible. Whereas, web-based software cannot be realized without architectural thinking to deploy into a large-scale application. Nevertheless, the researcher should investigate data-intense solutions like the operator assistant web-based software regarding robustness, scalability, high performance and security.

Architectural Pattern: The architectural patterns are used to connect different elements of the web-based software of web-based software to introduce a user experience to end users. The researcher will give details about the architectural patterns that could be thought to be related to practical implementation.

Monolithic Architectural Pattern: A monolithic architectural pattern can be composed in a single piece of software. In a monolithic architecture, the researcher cannot scale components differently, and interdependence 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 the researcher 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 a monolithic application 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 of an application 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 in a database with necessary configuration files distinctly. A significant advantage of a microservice decomposes an entire application into services. This architecture employs lightweight protocols, such as REST or Remote Procedure Calls [74].

Although microservices had advantages over other architectural patterns that have been mentioned in the section, there were several drawbacks that one should consider them. Adoption of microservices from different architectural patterns is hard since a system should be separated services and components to minimize in deploying as a distinct nature. 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 framework to increase deployment and performance features. Such features may cause more complicated architecture when the deployment of web-based software 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 will have a dependency from older versions. Older version issue creates an issue for future development. If there is a strong dependency between software components in an architecture, dependency problem can be enlarged, and the overall system may lose the feature of microservice out. The fault isolation is strong in microservices so that one component does not affect from an external failure which has been generated by another component.

As a consequence, the primary principle is dividing the component as much as possible unless the microservices 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 between components is as strong as monolithic architecture so that microservices can be distinguished from monolithic architectures.

Failures can affect multiple components because there might not be a clear separation between distributed components. Distributed components have not well-separated data storages and small units that may be scaled independently without having scaled entire architecture.

N-tier Architectural Pattern: Basically, the *N-tier* architecture comprises of the presentation tier, application tier, and business tier. Each tier can have multiple layers that can coherently work in a physical system. Though 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 that it is an organization of code. Layers can connect to each physical source via sockets and the 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. Operating systems, such as Linux and Windows, can transfer messages through shared memories. 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 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 data in permanent data storages. This architecture reduces the deployment costs and it is easy to optimize in the context of the code base. Unfortunately, it suffers from communication points as n-tier architecture increased communication points, which causes the complexity of communication environment. In addition, communication parts can share among application layer and business layer so that one can consider that it does not solve the 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." [75]. Each service discoverable addressed is called endpoints that transmits composed messages to each other so that supports the organization of distributed components in a black-box process [75]. This black-box process utilizes heavyweight XML-based protocols, such as SOAP or Web Service Definition Language. Despite the fact that it provides a decomposing way for an organization with a service provider, a service discovery and a service consumer, it is widely thought as a larger monolithic application. The SOA supports loose-coupling and service reusability as do

microservices, but there might be common data similarly in microservices; whereas, microservices discriminate data storages to regarding services.

Programming Design Pattern: This concept implies repeatable solutions for common problems that one encountered reoccurring issues. Even though there are numbered programming design patterns under several categories, the researcher will introduce some design patterns that have been argued 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 like models, views, and controllers to provide code reusing and independently deployment.

The model encapsulates business logic and business 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 layers of an application may concern changes that are presented by a model. The view demonstrates a view of the modeling of data submitted by the same data. It also strictly 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: It exemplifies that requests of workload can be conveyed between service providers and service requesters. Such servers can save states of sessions that have identified by clients or servers transmit information with clients in a stateless and cache-based way. Servers hold application logic to present information and calculation when a client requested. A client is responsible for transferring messages to end users since a client can initiate exchanging messages between servers and clients.

Moreover, a client can send a message or bulk data instantly to servers to check their health status whether a server is in service or out-of-order. This mechanism is called polling that can send a message from client to server within a particular time arrival. Two critical concepts differ in that respects. The synchronous communication refers to dependent communication individual messages in a blocking manner. Such messages should wait for the next messages to complete their operations. One task can execute an operation as soon as another task finishes. The 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 finished the operation. In this way, a user can send a request to a web service without waiting for the result of queries. Multi-thread management, finite state machine or event-based callback functions implement the asynchronous communication. As an example, event-based callback functions that take advantage of a scripting language which is a general type of asynchronous function.

Regardless of being benefited asynchronous or synchronous requests, a question might have been asked by readers. *"How can a web-based architecture adapt to increasing workload?"* The question leads us to a concept for distribution of workload using a medium, which is the so-called load balancer.

Single Page and Multi Page Applications: These two programming patterns are propitious for developing front-end applications. Single page applications refer to load and refresh all web elements in a web browser during use. Thus, it does not restore the entire page except for demanded content. The single page applications can cache local data so that one can load data in case of offline connection. When the offline connection turned out to be an online connection, data content is updated automatically. The multi page applications update the whole page upon each request to be exchanged between the server and web browser. Multi page applications are useful for data-driven applications especially like a search engine, question answering or database applications. Although they decrease the speed of communication between front-end and back-end frameworks, they increase 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 0.7: 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 0.8: Counterpart of sample CoffeeScript in Listing 0.7

B.4 Script Languages for User Interface Development

Fundamentally, script languages are a subset of programming languages. The front-end framework coherently works with script languages. Notwithstanding, a crucial step sets script languages apart programming languages, which is at the compilation level. Script languages are interpreting the source code instead of compiling it.

JavaScript: JavaScript is an entry point for the 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. The *ECMA Standard* is a de facto standard of JavaScript, and these properties affect other languages 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 the JavaScript language. The TypeScript can implement object-oriented paradigms such as interfaces, classes, abstract classes or objects. 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.

CoffeeScript: It is a similar script language to 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 it needs a step to compile from the CoffeeScript to the JavaScript. After conversion to CoffeeScript, it makes look complicated than Javascript because of pre-processor codes. Principally, the CoffeeScript reduces code complexity, but it makes a harder syntactic structure for a JavaScript Developer. While CoffeeScript is suitable for small module development that can easily be integrated into a more notable 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. The reader can see an example of CoffeeScript and compiled version with Javascript in Appendix B.2 and B.3.

PureScript: It is one of the most considerable cross-compiler support as compared to the previously mentioned three script languages. Besides, large scalability support is one of the fundamental features of PureScript so that it can be used in multiple operating systems, such as *C* and *Field Programmable Gate Array*. As compared to other script languages, PureScript has the most substantial scalability support for numerous platforms.

This script language takes the main features from the *Haskell Functional Programming Language*. PureScript supports 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*, PureScript handles with more complex *Object* structures. PureScript can modify *Classes* and instances with keywords with *"class"* and *"instance"* by creating advanced typed-data. The type definition is more enhanced than the JavaScript because the 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 pooling mechanism to use in case of need.

B.5 Dynamic Server (A quasi OPC-UA Server)

A publish/subscribe service defines the way of communication by using message-oriented architecture or standard transmission protocol of the *Open Systems Interconnection*. By utilizing a middleware, publisher and subscriber can be de-coupled. The subscriber or publisher may hold 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 are Broker-Based Middleware and Broker-less Middleware [76]. It is generally accepted that the broker-based middleware has a prevalent use in the 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 exploits message-oriented architecture, and it behaves as a publisher. Hence, it was designed differently from other OPC UA Server to takes messages that can be dispatched to particular receivers. Although the fundamental data collector is the *eniLINK* with sensors, the *"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 0.4, 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 the *eniLINK*. The *Message Broker Service* collects data from production and manufacturing system in the smart factory of Fraunhofer IWU in helping the creation of linked data.

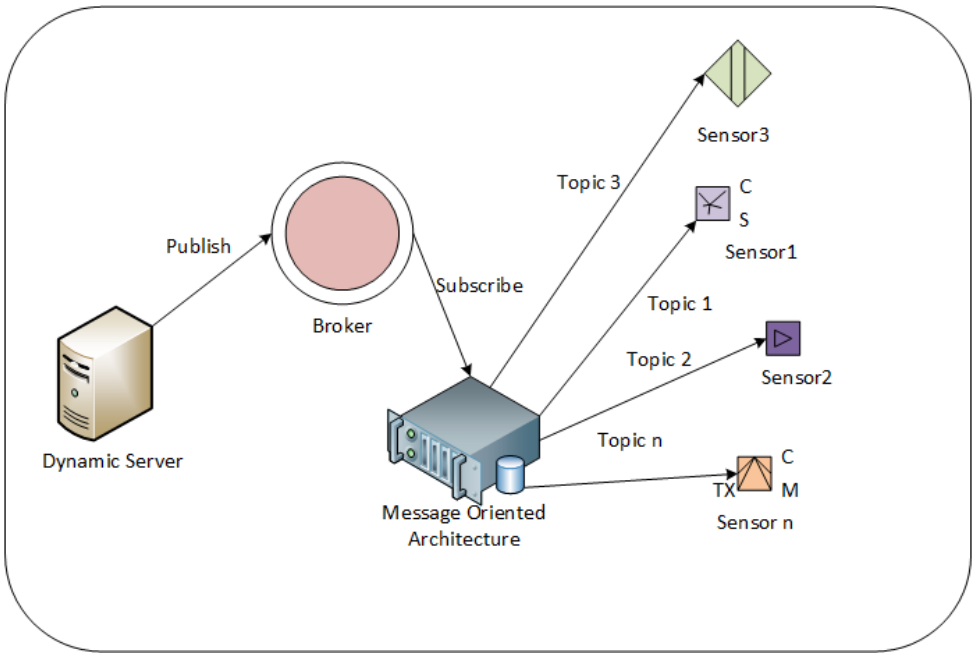

Figure 0.5: The Dynamic Server Publish/Subscribe Model

Types of Middleware	Advantages	Disadvantages
Broker-less Middleware	• No Central Point of Failure, • Legacy Devices Support • No additional software components like Broker	• Protocol Dependency because of non-existence a Broker

Broker-based Middleware	• Broker reduces latency and overhead generally	• A network bottleneck could be disastrous

Table 0.4: Types of Middleware Publish/Subscribe

B.6 OPC UA 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:      export.XSLTCaller()
6:  function Build Node Tree(nodes)                           ▷ Node Formatting
7:      client ← GetEndpoint()
8:      client ← Client(serverurl)
9:      global nodecumulated ←None
10:     nodeID ←0
11:     for node in nodes do
12:         nodecumulated = node.nodeid.Namespaceindex
13:         for ref < node.getreferences() do
14:             nodecumulated.extend( ref.nodeid.Namespaceindex)
15:         nodecumulated = list(set(nodecumulated))          ▷ Clear duplicates
16:     return nodecumulated                                  ▷ Return node id list
17: function Import Nodes(serverurl)                          ▷ Traverse Node
18:     client = Client(serverurl)
19:     client.connect()
20:     for ns < client.getNamespaces() do
21:         namespaces[client.getNamespaceIndex(ns)] = ns
22:     root = client.getRootNode()
23:     child = client.iterateChildNodes(root)                ▷ append child to node
24: function Export File(outputFile, namespaces = None)       ▷ Export into
    XML
25:     if namespaces != None then
26:         for node not in nodes do
27:             if node.nodeid.namespaceindex is namespaces then
28:                 nodes = [node]
29:             else
30:                 nodes = list(nodes)
31:
32:     export = XmlExport(client)
33:     export.BUILD NODE TREE(nodes)
34:     export.appendXML(outputFile)
```

Figure 0.6: Extraction Algorithm of OPC UA Address Space [69]

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 [66]

Latency Test: This parameter shows the round-trip time between two consecutive queries to be dispatched.

JSON Serialization Test: Through this test, JSON serialization has been made for the HTTP responses, and a key value was mapped onto the response of HTTP Requests as a value.

Platform Support: Platform dependency covers the leading platform or operating systems in the software world such as Windows OS, Unix OS, and Linux OS.

Multiple Query Test: In this test case, numerous rows of JSON data with *ID* and *random number* are sent per queries. This test case evaluates JSON serialization at multiple rows.

Single Query and PlainText Test: Plain text query sends a plain text message inside the body of the HTTP Request. This happens without caching, but the pipelining feature of HTTP is enabled [66]. Single Query test fetches data from a database and serializes it into the JSON format [66].

All percentages show the performance level of the different test case. The higher percentage of test cases would provide better achievement. Differently, lower latency shows better performance metric in the Table 0.5.

B.8 Frontend Framework Comparison

Feature	Angular 2+	React	Ember.js	MeteorJS	VueJS
Routing	Static Routing	Static Routing	Static Routing	Static Routing	Static Routing
Data binding	Two-way data binding	One-way data binding	One-way data binding	Template Binding	Two-way data binding
Feature Advantage	Object-oriented script development, Independent Library Dependency, Token Interceptor	The smallest possible component-based architecture	Code modularity	MongoDB based view-data storage	Advanced Dependency Injection Loosely coupled components

Dependent Pattern	Component-based	Compo-nent-based	Model-View-Model-Model	Model-View-Controller	Model-View-Model-View

Table 0.6: The front-end frameworks

Two Way Data Binding: Model part would be automatically updated upon alteration of data. This alteration commonly is provided by timeout intervals.

MVVM (Model-ViewModel-View): This architectural pattern provides separation of graphical user interface (separation of presentation logic). The controller contains main code base, and this holds a small amount of data state. Diversely from the Model-View-Controller, the MVVM does data binding between View and View/Model layers with interface functions of the View/Model.

Template Binding: It is a type of one-way data binding, and the element properties are binding with templates. The element that has assigned a template property is updating by one-way data binding principle.

One Way Data Binding: Model part does not update upon alteration of data. Components of the language do not control the state of data.

Static and Dynamic Routing: Static routing utilizes manual configuration to handle routing requests user defined hard-coded routing address. Dynamic routing takes automatic configuration assigned by network operators to handle routing requests.

B.9 Load Balancer & Reverse Proxy Configuration

```
http {
    proxy_connect_timeout 300s;
    proxy_read_timeout 300s;
    server {
        listen 80;
        access_log off;
        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;
        }
    }
    upstream backend {
```

Listing 0.9: 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
YiOjE1NDgzMzk1NDYsImV4cCI6MTU0ODk0NDM0NiwiaWF0IjoxNTQ4MzM5NTQ2fQ.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 0.10: 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 0.11: 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 0.12: Multiple Request with Load Balancing

B.13 Question Answering HTTP Request

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

Listing 0.13: Question Answering HTTP Request

B.14 Examples of Discovery Service

Application Name	Advantages	Disadvantages
OPC UA Local Discovery Server[14]	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

[14] http://opcfoundation.github.io/UA-LDS/

Node OPC UA Discovery Service[15]	Easy to use with web technologies Communication stack embedded into the node.js modules	Lack of automatically endpoint discover Realized as a desktop Application

Table 0.7: Comparison of the Practical Discovery Services

B.15 Screenshot of the Subscription Request

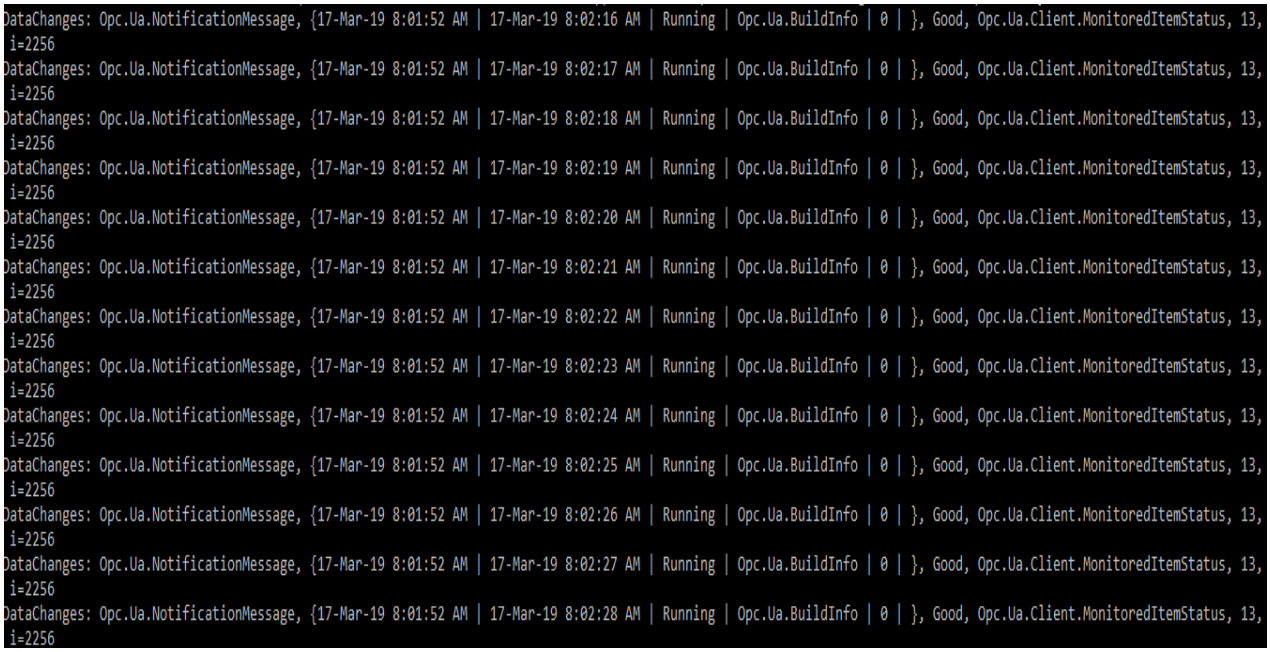

Figure 0.7: Subscription Request with a Monitoring Node

[15] https://github.com/node-opcua/node-opcua/tree/master/packages/node-opcua-local-discovery-server

B.16 Screenshot of the Local Discovery Server String Array

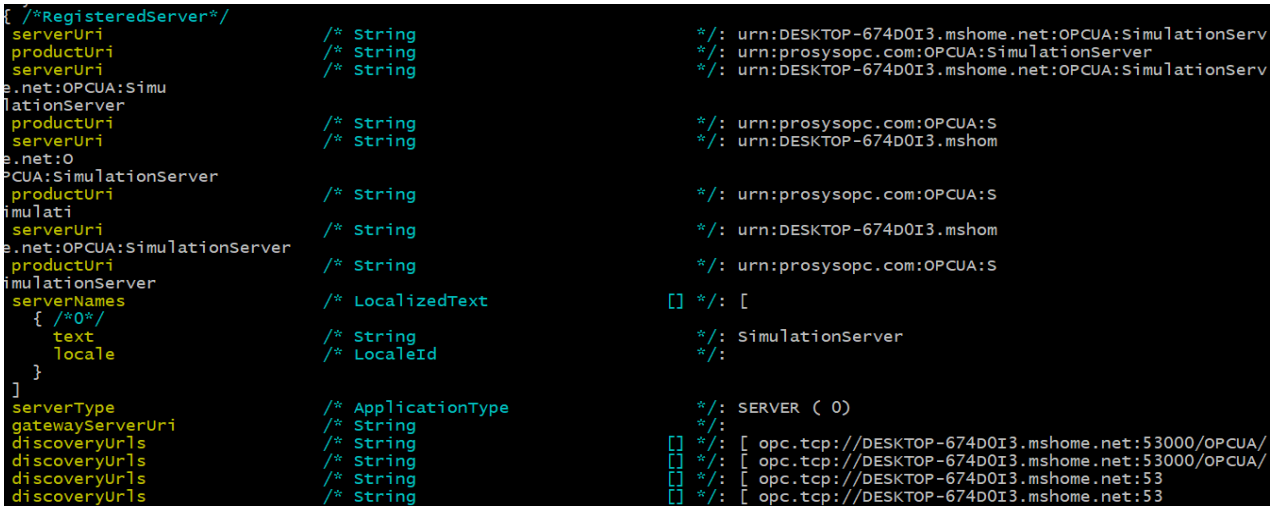

Figure 0.8: A snapshot from the local discovery server [77]

Appendix C Literature Review

C.1 Question Answering Summary

Study	Method	Content &Result
[Molla, Gonzalez Et al. 2007]	Literature Review on Restricted Domain for QA	Potential haphazard of restricted domain QA. Specifications of Restricted Domain QA
[Molla, Vicedo 2007]	Developed a particular framework instead of targeting specific domain	The authors concluded to port a solution into a framework could decrease time-consuming development tasks.
[Chung Et al. 2004]	Developed a QA system for SQL queries	Query frame can be mapped onto rule-based generated answer pattern
[Diefenbach Et al. 2017]	Published a survey paper about the Knowledge Base Question Answering	Very detailed analysis of the techniques of the knowledge-based question answering. The authors concluded to examine every existing technique is nearly impossible.
[Celikyilmaz 2006]	Implemented a probabilistic Bayesian Model	Briefly explained algorithm. Lack of evaluation and comparison of the algorithm.
[Giannone, Bellomaria & Basili 2013]	Examined the Hidden Markov Model to use unsupervised statistical learning.	The authors computed the probability for subsequent observables event from hidden states to implement an unsupervised method.

[Unger Et al. 2012]	Heuristic search with named-entity recognition and POS Tagger to match SPARQL onto natural expressions	Detection of properties with string similarity algorithms. The system does only care about the interchanged relationship between verb and nouns.
[Palaniappan, Sridevi & Subburaj 2018]	Created a QA by generating template-questions in the e-learning domain	Similarity matcher has been realized that aims at a different type of questions with different patterns which have to be matched.
[Ferre 2012]	Realized a controlled natural language	Specified the details of the linguistic features of the SQUALL.
[Luz, Finger 2018]	Developed a system that applies a recurrent neural network to transform natural language expressions into a SPARQL query	With neural attention method, there may be no need to extract linguistic features from complex grammar structures.
[Diekema, Yilmazel & Liddy 2004]	Criticized the development of test questions in a restricted domain	Defined parameters for performance testing, data source testing, and user interaction testing.
[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	Analyzation 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 0.8: Question Answering Literature Review Summary

C.2 OPC Unified Architecture in Web Environment

Study	Method	Content &Result
[Cavaliaeri, Salafia & Scroppo 2017, 2018]	Implementation of a back-end application for reading, monitoring nodes and writing request.	Loose-coupled architecture with a back-end practice. Detailed Publish/Subscribe Architecture analysis. No examination of balancing workload and front-end application aspect of generalization of the research.
[Paronen 2015]	Implementation and design practices about historical and direct data access. Detailed RESTful API calls with main functionality.	Several findings, which are: stateless protocol mapping onto stateful protocols, load balancing a specific session ID, polyglot development advantages, performance advancement with caching JSON serialized nodes.
[Grüner, Pfrommer & Palm 2015, 2016]	Implementation of a test application for various devices that range from industrial embedded devices to personal desktop computers.	Made an evaluation between various devices aspect of the session establishment and session integrity.

[Shiekofer, Scholz & Weyrich 2018]	Implemented very detailed HTTP methods with media types.	Emphasized main problems such as *HTTP Mapping*, *Session-less Invoke*, and *Browser Support*

Table 0.9: Literature Review OPC UA Web Component

C.3 Linked Data Collection for Heterogeneous Data Source

Study	Method	Content &Result
[Katti, Plociennik 2018]	Integrated OWL linked data language	Creating a semantically augmented OPC UA Framework that enhances knowledge in production for decision-making systems
[Pfrommer, Grüner & Goldschmidt Et al. 2016]	Realization of a technology independent common core model for information modeling	Creating a uniform information model that can adapt the information in the various data source (SQL DB, memory mapped value, etc.)
[Su Et al. 2014]	Offering a markup language to represent device parameters and measurements	Organization of a sensor markup language for real-time streaming data
[Wang, Zhang & Li 2015]	Defining rules for the semantic annotation tool	Automatic assignment tool for namespaces to be specified on the sensor and actuator applications
[Llanes Et. al. 2016]	Categorized the real-time data for linked-data with selecting ontologies, creating data linkages, choosing related datasets	Guidance with clear-cut defined steps for future research to apply real-time data into streaming linked data

[Anicic, Fodor 2011]	Proposed a language is called Event Processing SPARQL (EP-SPARQL)	Listed the main functionality of the language and emphasized the advantages in case of being used the language.
[Hasemann Et al. 2018]	Proposed an RDF tuple storage	Defined an RDF tuple storage that is fully compliant to SPARQL endpoints and HTTP methods.

Table 0.10: Literature Review for the Data Collection

Glossary

Inter-process Communication: It is a mechanism that allows the processes to communicate with each other and shares their messages through various methods such as shared memory, pipes, sockets and so on.

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

Reinforcement Learning: It is a kind of machine learning algorithm that allows software agents and machines to determine the ideal behavior automatically within a specific context, to maximize its performance.

Long Short Term Memory: LSTM is a unit of the 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 long-term bidirectional 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.

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 it is a 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 29. May 2019

_____

Orcun Oruc