Semantic Question Answering System for Smart Factories

Orcun Oruc, Adrian Singer, Ken Wenzel, *Fraunhofer IWU, IEEE*

Abstract Industrial production characterized by the increasing interconnection of machines and devices (e.g. Industrial Internet of Things, IIoT). Smart Factory evolved with their rapid requirements over the past few years. Leveraging Industry 4.0, smart factories require Human Machine Interface solutions in order to present information to experts, users or operators. Semantically created data by manufacturing machine mostly used by Human Machine Interface System. As a result, an information model can be extracted from an extensible markup language in order to use with semantic data. Nature of linked data might has different properties from other open domain or closed domain question answering so that we will examine a restricted domain question answering system with linked data through our main architecture. Moreover, we present a semantic question answering with data based on eniLINK and a generated data from an OPC UA Server known as Dynamic Server. In addition, we will evaluate our question answering with different measurements according to the requirements of a restricted-domain question answering.

***Index Terms*— Semantic Web, Web 2.0, Web Services, Information Retrieval**

I. INTRODUCTION

THIS work introduces a new concept for smart factories in terms of HMI technology integrated into a semantic question answering. A semantic question answering used for information retrieval to provide answers from questions by means of linked data. Linked Data is a new concept ubiquitously covered all around of use. The data size, which resides in the Web has been doubled every year since the day of the Web invented. As a result, a great amount of unlabeled data could not be used by applications, so W3C (World Wide Web Consortium) decided to create a standard for Semantic Web in order to create linked open data concept.

Requirement 1: Linked Data Generation from a Fraunhofer IWU's OPC UA Server,
Linkedfactory Static and Dynamic Data created by KVIN Service with LevelDB (key-value storage library).
Data source is a key factor for a successful question answering system. Our question answering uses linked data created by eniLINK.
Requirement 2: Real-time linked data generation through KVIN API.
Requirement 3: A standard-based interface compatible with RESTful communication.
Requirement 5: Evaluation of Semantic Question Answering
The remainder of this paper is organized as follows: Section 2 describes requirement and approaches, Section 3 defines how we can prepare real-time data. In Section 4, we clearly explain the prerequisite methods in natural language processing. Thereafter, Section 5 clearly examines the proposed architecture. As a result, we conclude in Section 9.

II. REQUIREMENT AND APPROACHES

//In this section you should elaborate the requirements

III. DATA PREPARATION

A. Mapping a OPC UA Data into a Semantic Data

Our main data sources are semantically parsed data from eniLINK [1] and OPC UA Server generated data. In the phase of OPC UA Server generated data, we used a SDK which is published by FreeOPCUA [2]. OPC UA Protocol uses an information model and the information model can be used with metadata to simulate in other OPC UA Server with languages such as XML (Extensible Markup Language). Due to the nature of XML, it is a language a strong hierarchical and hardly extendable. However, semantic data such as Resource Description Framework (RDF) can employ triples with SPARQL. Different RDF Graphs stored in eniLINK[1] are uniquely identified .

Algorithm 1 Node Extraction

```
function MainFunction()                                   ▷ Starting point
    export = ServerExport(serverurl, filename)
    export.IMPORT NODES(serverurl)
    export.EXPORT FILE(outputFile, namespaces)
function Build Node Tree(nodes)                           ▷ Node Formatting
    client ← GetEndpoint()
    client ← Client(serverurl)
    nodecumulated ←None
    nodeID ←0
    for node < nodes do
        nodecumulated = node.nodeid.Namespaceindex
        for ref < node.getreferences() do
            nodecumulated.extend( ref.nodeid.Namespaceindex)
        nodecumulated = list(set(nodecumulated)   ▷ Clear duplicates
    return nodeID                                 ▷ Return node id list
function Import Nodes(serverurl)                          ▷ Traverse Node
    client = Client(serverurl)
    client.connect()
    for ns < client.getNamespaces() do
        namespaces[client.getNamespaceIndex(ns)] = ns
    root = client.getRootNode()
    child = client.iterateChildNodes()
function Export File(outputFile, namespaces = None)   ▷ Export into XML
    if namespaces != None then
        for node != nodes do
            if node.nodeid.namespaceindex is namespaces
                nodes = [node]
            else
                nodes = list(nodes)

        export = XmlExport(client) then
        export.BUILD NODE(nodes)
        export.appendXML(outputFile)
```

Figure III-1. OPC UA Address Space Representation in XML [2] [3] **.**

A. Real Time Data Mapping with KVIN

Real time data has intricacies to use as linked data taken from sensors, actuators or software logs. In the aspect of Smart Factories, real-time data mostly is created by sensors and actuators that are underlying structure of manufacturing machines. Fraunhofer IWU collects the real data source by saving into a time series database. The major drawback is that when a time series data taken, semantic query endpoint can not use the pure data without annotating. Such annotation could be triple creation, adding of predicates or serialization from one formal language to another.
Our architecture is providing a real time semantic data annotator KVIN that utilize to extract triples from time-series data in a database.

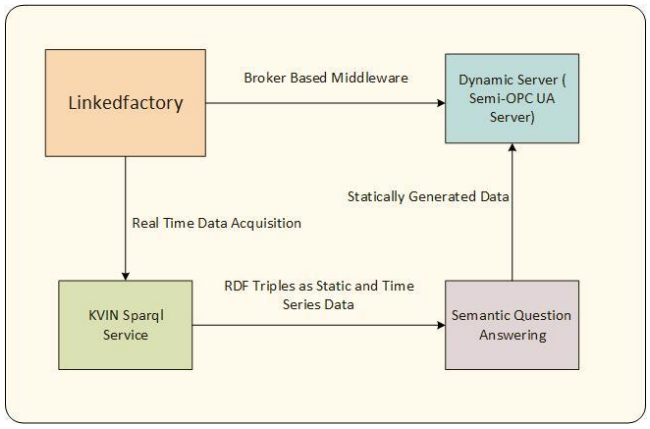

Figure III-2. KVIN Continuous SPARQL Mapping.

This work proposes a service named KVIN to perform a SPARQL request into a specified endpoint. This service is based on a combination of triple store through Level DB which is a key-value storage library written by Google [1]. It has been used a RDF4J's extension to create a SPARQL Service.

IV. NATURAL LANGUAGE UNDESTANDING FOR THE SEMANTIC QUESTION ANSWERING

In Natural Language Processing, we need to identify sentence's structure in order to reach at the step named Query Formulation of SPARQL.

Preprocessing: Chiefly, all natural language tasks start with preprocessing which means cleaning data for specific tasks. This could be reduction of undervalue data, reduction of discrepancies between values or removing non-related morphological properties.

Tokenization: A question answering system should parse all input as tokens. Tokenization is an initial step for Part of Speech tagging to parse into verb, noun, cardinal numbers, adjective etc.

Lemmatization and Stemming: Lemmatization and Stemming are similar steps to each other with one difference. While stemming used to find syntactical structures, lemmatization looks for a semantic structure. Stemming clear of the structure of suffix and prefixes. In our system we are supposed to use a lemmatizer and stemmer in order to reduce lexical complexities. A lemmatizer has been used to consider about morphogical analysis of verbs, e.g. from "contains" or "contained" to "contain". Then we use this verb mapping into a predicate to construct a SPARQL Query. The lemmatization and stemming are part of normalization process in terms of linguistic properties.

Part of Speech Tagging: It is a preprocess step for parse tree to identify item tagger such as verbs, adjectives or nouns. 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 [4]. Part of Speech Tagger mostly uses a Markov Model that is a part of statistical natural language understanding. Markov Model stands for a state can be 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 defines its neighbors, e.g. nouns are preceded by determiners, adjectives, verbs [4]. For example, a chess player makes a movement according to the last movement of a rival rather than guessing from the first movement of the rival. In this step, pre-saved corpora which has million words has to be tagged by POS Taggers.

[1] https://github.com/google/leveldb

Parse Tree and Penn Treebank: One of the common list that has identifier for POS denominated as Penn Treebank. A treebank used for annotating syntactic and semantic structure of a sentence with million words of part-of-speech tagged text.
When a natural query is given, a question answering system should understand the grammar behind of it. POS tagger is not enough to identify grammatical structure for complex natural queries. Relationships among noun phrases, adjective phrases, adverb phrases, and verb phrases should be examined in order to map correctly subject-predicate-object triples in linked data. The approach of parsing separated into two main sections, which are the rule-based approach and the probabilistic approach [5]. The rule-based approach is a top-down approach to solve problems via predefined rules such as Regex-parsing. Therefore, a question answering system should define rules precisely to get a correct answer. Open-domain question answering systems use this approach because of complexity of bottom-up approach and broadened question types. Nevertheless, rule-based approach could give undesirable results in restricted-domain question answering or semantic question answering and could be time-wasting approach. A dependency parser analyzes the grammatical structure of a sentence and it gives the relationship among them. The dependency parser also give the relationship between general words and root words. Thus, we can identify the center verb or noun from complex sentences. This parser utilizes a dependency treebank file and word embedding files. Chiefly, a dependency parser applies supervised machine learning method to reach syntactical result. For example, with dependency treebank, data is broken into test and training set, however, word embedding used for training phase.

Dependency Parser: //Explain
Constituency Parser: //Explain

Named Entity Recognition: It is a subtask of information extraction to locate and classify named entities with pre-classified labels such as names of people, organizations, locations, and quantities etc. Named-entity recognition is a method that identifies the item of a sentence as a domain-specific. It identifies all structures mainly as a person, a location, an organization, and an entity.

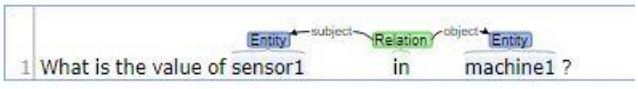

Figure IV-1. Natural Language Processing Steps for Question Answering.

As shown Figure V-1, “sensor1” and “machine1” named as entity and found a relation between each other. After doing a couple of experiment on named-entity recognition, evidences show us a 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. In order to create a named-entity recognition for a smart factory, a model can be trained satisfying the requirements of a smart factory. In this context, a model can be created by statistical methods or rule-based model. In context with rule-based model, character regex method can identify the structure of a natural query. For instance, a named-entity recognizer can employ a model that contains combination of “HeatMeter” or “HeatingWater”, which it can assort given items with the character started by “Heat” in a smart factory.

V. PROPOSED ARCHITECTURE

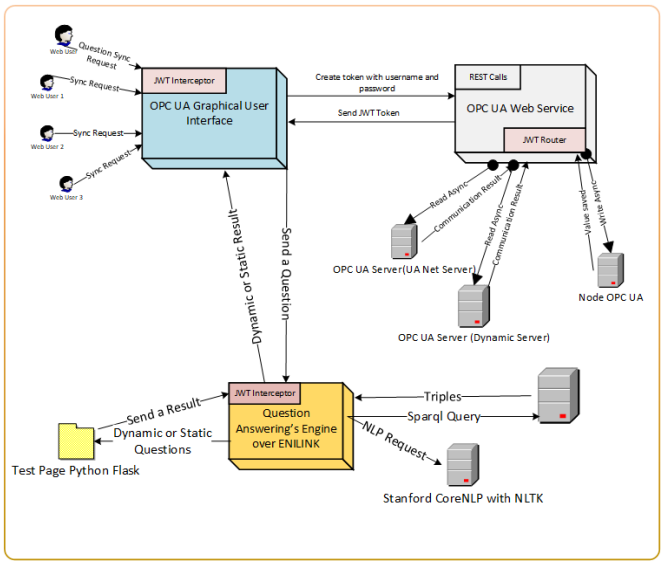

Figure V-1. A RESTful Architecture for Question Answering.

As shown above in Figure 1, we aimed a RESTful architecture for the semantic question answering as an orange box. NLP Requests are sent to Stanford CoreNLP server to parse grammatical structure on given the natural query. Our semantic question answering system handles with token normalization process with Tokenization, Lemmatization and Stemming

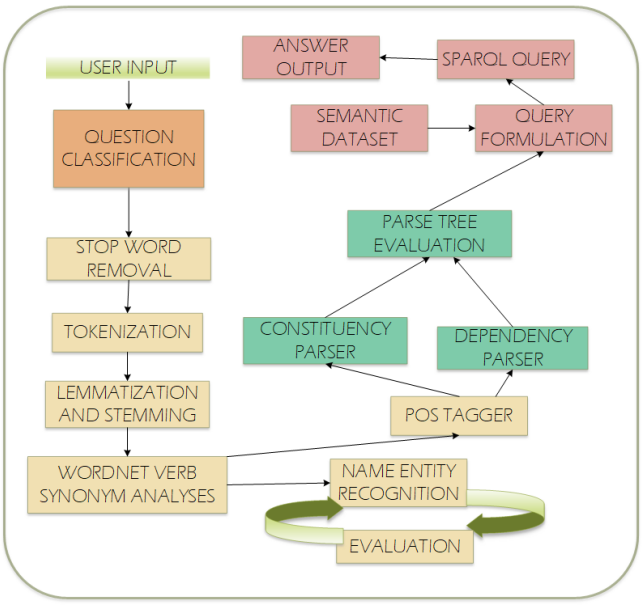

Figure V-2. Natural Language Processing Steps for Question Answering.

As illustrated in Figure 1. , we have multiple stages to get an answer at the end. Restricted domain question answering suffers from a problem of name-entity recognition when we implement the system.

After taking an input from any user, stop-word preprocessing stage start to filter unnecessary characters such as question mark, exclamation point, comma, dot or determiners. Tokenization is the next step in order to reduce the size of characters to provide an optimization in natural language processing and it reduces complexity of instances of sequence characters. Lemmatization and Stemming is an essential step before WordNet Verb analysis because our main target is to extract verb, nouns and related chunking in order to formulate a SPARQL query that can give an answer.

Algorithm 1 Query Formulation

```
function QUERY FORMULATION(a, b)                          ▷ Explain here
    query ← QueryWithPrefixes
    r ← contituent.parse.tree
    indirectdependency ← dependency.parse.tree
    while nodes ≠ leafs.terminal do                ▷ Until leaf nodes(Terminals)
        verbs ← PARSER(nodes)
        nouns ← PARSER(nodes)
        similarityflag ← WORDLATENANALYSIS(verbs)
        if StaticInformation is True then
            indirectdependencyFlag ← DEPENDENCYPARSER(nodes)
            if similarityflag and IndirectDependency is true then
                object ←nouns
                predicate ←verbs
                query += object + predicate + ?subject
            else
                subject ←nouns
                predicate ←verbs
                query += ?object + predicate + subject
        if DynamicInformation is True then
            predicate ← PARSER(nodes)
            object ← PARSER(nodes)
            similarityflag ← SIMILARITYLEVENSHTEIN(input)
            query += object + predicate + ?subject
    return query                                ▷ The last query has been constructed
```

Figure V-3. Natural Language Processing Steps for Question Answering.

The general information about our architecture is in Figure 1.1 below: The RDF data from eniLINK and OPC UA Server (right-hand side). A SPARQL Endpoint has been provided by our architecture for local static data and KVIN also presents a SPARQL Endpoint for time-series data.

VI. EXPERIMENTAL DEVELOPMENT

A. *Data Sets*

In the evaluation phase, our main data sources are OPC UA Server Generated Data, eniLINK Static, and Dynamic Data. Chapter 3 A. and B. parts are examining in a detailed way how to create and obtain data and convert into a linked data format. OPC UA Generated Data has not specific namespace definition unless we define as we wish. However, user defined IRIs definition has drawbacks such as collision or non-extendibility. A long listed linked data makes the structure complex so that two subject of list can be collapsed because of same-defined IRIs. In our case, all namespaces are generated with http://www.example.org" or "<unknown_namespace>".

One of the poorest performance has been observed in the phase of named-entity recognition because restricted-domain requires different lexicons to train entities.

B. *Model Setup*

VII. EVALUATION

Evaluation criteria comprises of question classification and recall, accuracy, precision, F1 score of answers against semantic question answering system. General evaluation parameters for a restricted domain question answering is not only limited with answering of questions, but also we can assess with speed, user interaction, querying style (keywords, browsing, spell checker, abbreviation recognition)

Give a result from Question Types as tables
Who, Which, How, Affirmation, Unknown

Give a result from Question Classification – both what type questions and TREC labeling

QUESTION CLASSIFICATION- QUESTION LABELS

Question Classification	Precision	Recall	Accuracy
What	A	B	C
Who	A	B	C
Which	A	B	C

How-Why	A	B	C
How many-quantity	A	B	C
Affirmation	A	B	C
Unknown	A	B	C
Overall	A	B	C

Li&Roth Taxonomy	Precision	Recall	Accuracy
Abbreviation	A	B	C
Description	A	B	C
Entity	A	B	C
Human	A	B	C
Location	A	B	C
Numeric	A	B	C
Overall	A	B	C

QUESTION ANSWERING RESULT

Question Answering Parameters	Total Questions
Correct Answer	A
Wrong Answer	A
Precision	A
Recall	A
F1	A
X	A
X	A

VIII. RELATED WORKS

[Diego Molla et. al] reviewed a main characteristic of question answering in restricted domains is the integration of domain-specific information that is either developed for question answering or disclosed for other purposes [6]. [Diego Molla, Jose Luis Vicedo] defined main characteristics of question answering system over limited domains, e.g. circumscription of question answering, complexity of question answering, and practical usage of question answering[6].

The authors have compared between open-domain and restricted-domain question answering by figuring out key points. [Diego Molla, Jose Luis Vicedo] offers four clear-cut subjects such as the size of data, domain context, resources, and use of domain-specific resources.

[Sebastian Ferre] has published one of the detailed reports that expresses common pitfalls of natural language processing, essential points while consolidating SPARQL query and morphological definitions [7]. 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 complexities [7]. It has grouped all substantial features of a morphological language and pointed out what type of features in a natural language harnesses with regarding priorities and orders. Main contribution of SQUALL is categorizing ambiguities of natural languages and how turned out an advantage when using a controlled natural language [7]. The authors sketched a translation from their intermediate language to SPARQL to gain more accuracy with their system [7].

IX. CONCLUSION

Please include a brief summary of the possible question answering implications of the work conducted at Fraunhofer IWU in this section.

APPENDIX

Appendixes, if needed, appear before the acknowledgment.

ACKNOWLEDGEMENT

This work is supported by Fraunhofer IWU. We would like to thank to group of HMMI on the account of financial support.

REFERENCES

[1] L. D. Platform and F. IWU, ‘eniLink’. [Online]. Available: http://platform.enilink.net/. [Accessed: 23-Nov-2018].

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

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

[4] 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.

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

[6] D. Mollá and J. L. Vicedo, ‘Question answering in restricted domains: An overview’, *Comput. Linguist.*, vol. 33, no. 1, pp. 41–61, 2007.

[7] 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.