

A HYBRID APPROACH FOR DEVELOPING, EXTENDING, AND IMPLEMENTING
INDUSTRIAL MAINTENANCE KNOWLEDGE GRAPHS AND SEMANTIC
ONTOLOGIES TO SUPPORT SMART MAINTENANCE DIAGNOSTICS

by

Renita Tahsin, M.S.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Engineering Management
December 2022

Committee Members:

Farhad Ameri, Chair

Meysam Khaleghian

Dincer Konur

COPYRIGHT

by

Renita Tahsin

2022

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

This work is protected by the Copyright Laws of the United States (Public Law 94-553, section 107). Consistent with fair use as defined in the Copyright Laws, brief quotations from this material are allowed with proper acknowledgement. Use of this material for financial gain without the author's express written permission is not allowed.

Duplication Permission

As the copyright holder of this work I, Renita Tahsin, authorize duplication of this work, in whole or in part, for educational or scholarly purposes only.

ACKNOWLEDGEMENTS

First, I would like to express profound appreciation to my thesis advisor Dr. Farhad Ameri who supported me at a time when I was sorely in need of it. Not only has he provided an opportunity to diversify my career from core industrial engineering to data analytics, but also did he guide me in each step of these two years journey. His door was always open whenever I had trouble with my research project or adjusting to other courses. His vast expertise in Ontology Engineering and data-driven decision-making pushed me to expand my thought process. In the end, I am amazed to see how much I have learned to deal with data and successfully implemented it in the thesis.

Second, I am thankful to my other committee members, who were incredibly supportive from the very beginning. I have taken course works with all of my committee members and am indebted that those course works are the reason I am working as a full time healthcare system engineer even before graduating with this second masters. I am whole heartedly grateful to my committee members and absolutely fascinated to be a part of this wonderful institution.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	vii
LIST OF FIGURES	viii
LIST OF ABBREVIATIONS	xi
ABSTRACT	xii
CHAPTER	
I. INTRODUCTION	1
Problem Statement	2
Assumptions, Limitations, and Delimitations	3
Assumption	3
Limitations	3
Delimitations	4
Research Questions	5
Research Methods	5
Research Plan	10
Task 1: Creating and Extending Maintenance Thesaurus	10
Task 2: Creating and Extending a Formal Ontology	11
Task 3: Knowledge Generation and Expansion	11
Task 4: Validation of the Developed Semantic Models	11
II. LITERATURE REVIEW AND SEMANTIC DEFINITIONS	13
Linguistic and Grammatical Maintenance Text Analysis	27
Semantic Technology Definitions	30
Simple Knowledge Organization System (SKOS)	31
Resource Description Framework (RDF)	33
RDF Triples Structure	34
RDF Graph	35
OWL Ontology	38

III. METHODOLOGY FRAMEWORK	40
Task 1: Thesaurus Development.....	41
SKOS Elements	42
Concepts.....	42
Labels.....	43
Semantic relationships.....	44
Mapping properties.....	45
Concept collection.....	45
SKOS Tool Functions.....	45
Nestor Tool Experimentation.....	48
Task 2: MWOO Ontology	50
Building OWL Ontology.....	51
Individuals.....	54
Properties.....	55
Classes.....	56
OWL Ontology Definitions.....	57
Owl Ontology Visuals.....	59
Task 3: Knowledge Graph Generation	63
Example 1.....	66
Example 2.....	68
Example 3.....	71
Task 4: Validation.....	73
Query Examples.....	77
Query Example 1.....	77
Query Example 2.....	80
Query Example 3.....	81
Query Example 4.....	82
Query Example 5.....	83
Query Example 6.....	85
Query Example 7.....	86
Query Example 8.....	87
Query Example 9.....	89
Validation Using Reasoner.....	91
IV. CONCLUSION.....	94
REFERENCES	101

LIST OF TABLES

Table	Page
1. The Timetable of Tasks to Accomplish the Goal	12
2. Test Maintenance Company Example Problem Statements	28
3. Test Data Example with RDF Triple Structure.....	35
4. Test Maintenance Raw Text Input and Subsequent Outputs Identified by Nestor	50
5. Natural Language Definition of MWOO Ontology Classes	57
6. Natural Language Definition of MWOO Object Properties	58
7. Examples of State and Process in MWOO Ontology	62

LIST OF FIGURES

Figure	Page
1. Few Examples of the Raw Maintenance Work Orders.....	6
2. Thesaurus and NLP for Hybridized Data Classification.....	7
3. Owl Ontology Classes and Object Properties in Protégé.....	8
4. Visualization of Single Raw MWO.	9
5. What is Observed vs. What is Diagnosed.	30
6. SKOS Model Example for Leaking.	33
7. Test Data Example with RDF Graph.	36
8. Use of RDF-type and RDF-label in SPARQL.	38
9. Methodology Framework from Start-to-End.	41
10. Partial View of User Interface in the Test Maintenance Thesaurus.	44
11. Partial View of Java Tool Showing the Semantic Relationship in Thesaurus.....	45
12. The Opening View of the INFONEER SKOS Tool.	46
13. Term Selector View with Example Raw Text in SKOS Tool.	47
14. Entity Extractor View with Example Raw Text in SKOS Tool.	48
15. Partial View of Nestor Tool used for Data Tagging.	49
16. SKOS Thesaurus Concept Mapping in Protégé Owl Ontology.....	51
17. Protégé MWOO Ontology Class View.....	52
18. Protégé MWOO Properties View.	54
19. Example of Artifact Instances in the MWOO.....	55

20. Control Box Property Relations in the MWOO Ontology.....	56
21. Portion of Material Class Annotation and Description in MWOO.....	57
22. Simple Examples of Ontological Relations in MWOO.....	60
23. Class State vs Class Process in MWOO.	61
24. Complex Example of Ontological Relations in MWOO Ontology.	63
25. Maintenance Workorder Annotation Tool (MWOAT) Tool Full View.....	64
26. Example 1 Analysis in MWOAT Tool.	67
27. Example 1 Knowledge Graph Visualization in RDF Grapher.	68
28. Example 2 Analysis in MWOAT Tool.	69
29. Example 2 Knowledge Graph Visualization in RDF Grapher.	70
30. Example 2 Analysis in MWOAT Tool.	71
31. Example 3 Knowledge Graph Visualization in RDF Grapher.....	72
32. Partial View of Star dog Studio Query Application.	75
33. Simplest Query Structure using SELECT Command.....	76
34. Query Example 1.	77
35. Query Example 1 Result Partial Visual Representation.	78
36. Query Example 1 with LIMIT Command.	80
37. Query Example 2 and Partial Result Visual Presentation.....	81
38. Query Example 3 and Partial Result Visual Presentation.....	82
39. Query Example 4 and Partial Result Visual Presentation.....	83
40. Query Example 5 and Validation from Ontology.....	84

41. Query Example 6 and Partial Result Visual Presentation.....	86
42. Query Example 7 Comparison.....	87
43. Query Example 8A.	88
44. Query Example 8B.....	89
45. Query Example 9.	90
46. Asserted Relations without Reasoner.	92
47. Inferred Relations with Reasoner.....	93
48. Instance Count per Class in Ontology.	97
49. Raw Text and Knowledge Graph Comparison.	98

LIST OF ABBREVIATIONS

Abbreviation	Description
CMMS	Computerized Maintenance Management Systems
SKOS	Simple Knowledge Organization Systems
OWL	Web Ontology Language
MWO	Maintenance Work Order
NIST	National Institute of Science and Technology
NLP	Natural Language Processing
RDF	Resource Development Framework
W3C	World Wide Web Consortium
URIs	Uniform Resource Identifiers
URLs	Uniform Resource Locators
IRI	Internationalized Resource Identifier

ABSTRACT

The unstructured historical data stored in Computerized Maintenance Management Systems (CMMS) is a mine of maintenance diagnostic information. This data is often underused due to its unstructured and informal nature. This thesis will propose a framework for transforming maintenance log data, which is often in the form of natural language text, into formal knowledge graphs. The proposed method generates a knowledge graph that encodes the semantic relationships between multiple maintenance entities based on the historical data that can be found in maintenance work orders. The knowledge graph is created semi-automatically through the hybrid application of text analytics techniques and human-assisted semantic tagging of maintenance work order text. The semantics of the knowledge graph proposed in this research will be provided jointly by a SKOS thesaurus and an OWL ontology. SKOS (Simple Knowledge Organization System) and OWL (Web Ontology Language) are both Semantic Web standards that will enhance the reusability and portability of the final knowledge graph. The knowledge graph created as an output of a java based tool will become an open-source shared industrial maintenance knowledge base that can be extended incrementally and be used for various decision support applications, including maintenance diagnostics and root-cause analysis. An online knowledge graph platform will be used to conduct querying and inferencing over the graph to support smart maintenance diagnosis.

Keywords: knowledge graph, thesaurus, Natural Language Processing, ontology

I. INTRODUCTION

Maintenance is the process of ensuring that machines and equipment operate continuously and efficiently with reduced breakdown or malfunctioning. In the absence of a maintenance management system, companies usually follow the corrective maintenance strategy, which means they use their machines and equipment until they fail, then repair and restore them when they can no longer function. However, downtime is a critical issue that has a direct impact on a company's profit. To reduce downtime and extend the life of assets, preventive, corrective, predictive, and periodic maintenance procedures are often implemented jointly. These procedures are often supported by software systems that enable maintenance technicians to plan their activities and to record details about the nature of failures, their probable causes, and the recommended treatment for the observed failure. These data are stored in large databases of software systems like Computerized Maintenance Management systems (CMMS) or ERP solutions.

This thesis focuses on the unstructured text-based raw data in Maintenance Work Order (MWO) that are often stored in CMMS databases. MWOs contain significant details such as problem statements, asset information and failures, the type of maintenance done, along with their scheduling, and the treatment used to address the observed problem. The valuable data stored in CMMS databases are often underutilized because they are in natural language format, which has several deficiencies, including ambiguity, incompleteness, and informality. The objective of this thesis is to convert maintenance work order data into a more computationally available format and enable the reusability of the knowledge embedded in maintenance logs. For this purpose, a framework supported by Semantic Web standards will be proposed for converting raw text into a

graph-based model, allowing historical maintenance information hidden in the text to be revealed, easily interpreted, organized, and reused. Our thesis focus is on industrial maintenance data, but the proposed method is applicable to other domains, such as healthcare and biomedical practices, where vast amounts of textual data need to be processed. We have conducted this thesis research in collaboration with the National Institute of Standards and Technology (NIST). NIST researchers have provided us with the necessary raw data and valuable information on the real-time picture of how the industries use maintenance findings from their stored database. We will briefly illustrate our vision of developing an industrial maintenance ontology enabling data-driven discoveries to enhance smart maintenance.

Problem Statement

Even though maintenance databases CMMS have gotten more structured and mature over time among researchers and companies, they still contain a significant amount of unstructured data that is difficult to identify and use. Not only is the maintenance data diverse across multiple industries, but operations also documented by maintenance technicians often presume that the data does not need to be explicitly represented. As a result, the MWO is often full of technical jargon and overloaded meaning. Besides, the domain knowledge to manage and reuse the heterogeneous maintenance data is a complex process and different for individual companies. Big to mid-size companies oftentimes are reluctant to share their database. Nevertheless, it is still possible in many domains to formalize the maintenance knowledge and share it among the general public. Another big problem is achieving a total collaboration throughout the data, semantic and application layers to retain the desired output

knowledge graph and ontology since the progression from raw data to ontology is tedious and often cumbersome, resulting quickly in a productivity bottleneck. Hence, the main objective of this thesis is to introduce a hybrid methodology for generating machine actionable knowledge from unstructured raw data and constructing an open-source knowledge graph as well as maintenance ontology.

Assumptions, Limitations, and Delimitations

Assumptions

- Our developed framework for semantic annotation will keep humans in the loop for the 1st level tagging and construction of the three levels of data layers (data, semantic, application) which includes thesaurus development for the manufacturing company-specific data. From the literature review, we have seen that maintenance operations are not always explainable, and human intuitions play a big role in formalizing knowledge.
- It is difficult for standard NLP methods to parse through the engineering work order jargon, and so we have to rely on the NLP tool for the 2nd level tagging of single and multi-phrase words
- We have no control over how the data is entered in CMMS and collected. We work with the raw data without changing its structure or giving instructions to operators as to how to enter data.
- The framework will reuse some of the existing semantic models (thesaurus and ontology) that have been developed previously.

Limitations

- We are limited by the availability of datasets provided by a few companies.

- We do not have access to maintenance technicians from the companies that have provided us with data. We need to interpret the meaning of terms and phrases ourselves based on our previous knowledge of maintenance.
- Every company produces valuable data, and often there are existing tools the management uses to extract the data. Nevertheless, the question is whether all the information available has been documented in the most beneficial way and whether this data has been used for fact-based maintenance management. It is similar to our circumstances since we are working with heterogeneous data across the industries collected by the maintenance technician, who often presumes it as part of a priori knowledge. But it is still possible to categorize the raw data to improve the reusability and findability of the knowledge embedded in MWO.
- Sometimes the problem statements do not have enough information regarding the failure or the reason and location of the failure. In those cases, human intervention is needed to decide on the implicit meaning of the text or just ignore them.
- Since our research topic is comparatively novel, it required suitable tools to be developed. As we find new words in the raw data that are not bucketed under the concept, we make the corresponding updates in the thesaurus to accommodate the new terms.

Delimitations

- We are only using input data that is generated by CMMS in CSV format. We are not using other maintenance documents such as standards, procedures, and reports in this work.
- We are only mapping the failures and associated relations with the items, not the

solutions. Solution could be the future addition to this research work.

- Currently, we are only using free software to analyze our data, such as Protégé and Nestor. Besides, we have also developed our own java-based tools to utilize the SKOS and ontology output as well as run the SPARQL query.

Research Questions

- What main concept categories and sub-categories can be used for classifying the key terms that appear in MWO data?
- Does using tools with NLP support (such as Nestor) improve the efficiency of the tagging process?
- How to categorize the Nestor-tagged problems under the correct category/bucket in the manual thesaurus?
- How effective will our final model show the semantic relationship among various entities?
- What should be the top-level concepts of the MD thesaurus?
- How would we express the queries to satisfy our competency questions, such as:
What is the cause of this maintenance problem, where is the location of the problem, what are the WOs related to this problem (s), and so on?
- How to use rules to expand the knowledge graph automatically?

Research Methods

This study focuses on converting the raw maintenance log into a formal knowledge graph that encodes semantic relationships among multiple maintenance entities. Figure 1 is an example of the manufacturing company's raw maintenance data we will be using to generate the knowledge graph.

Random_ID	Problem_Statement
1497B239-42A	CALL FOR THE UNIT NOT STARTING. FOUND DEAD BATTERY
53E3A449-E1	CALL FOR UNIT NOT CHARGING- FOUND THE CHARGER FAILED
61BE1ADC-F	o-ring for boss fitting at drive motor has ruptured
B49D4E42-8	GEN NOT WORKING, KEN W/ GENIE ADVISED TO CHANGE RESISTOR AND REPLACE POT SWITCH INSIDE BOX.
F0AE6037-1E	engine stalling; see Claim 522888.0 Needed to replace fuses
509279A5-6E	UNIT IS NOT WORKING FOUND FAULTY LIMIT SWITCH
097F59CF-C	EGR TUBE BROKEN AT FLANGE.
80319764-D	no platform controls bad control board
D9A76162-0	FRONT DIFF IS MAKING OIL, FRONT DIFF BLEW OUT SNAP RING INTERNALLY, REMOVED ONE SIDE AND FOUND DIFF TO BE NO GOOD SNAP RING BLEW APART CAUSING DAMAGE TO CASE.
66D0521C-F	oil leak o ring on main control valve
	fuel pump float is bad stuck doesn't work

Figure 1. Few Examples of the Raw Maintenance Work Orders.

The methodology that will be used in this research leverages the existing Semantic Web standards for knowledge and data representation, including OWL, SKOS, and RDF. As mentioned in the introduction section, our thesis starts with collecting the raw maintenance data stored in CMMS and building a maintenance thesaurus or SKOS model. SKOS is a standard, published and recommended by World Wide Web Consortium (W3C), that provides a more organized framework for building controlled vocabularies such as thesauri, concept schemes, and taxonomies to be used and understood by both human and machine agents. Besides, SKOS is in a stable, standardized state; therefore, it can be incrementally and modularly extended by linking its concepts to external concepts from other graphs if needed in the future. This SKOS thesaurus will provide us with a formal vocabulary of maintenance terms and will be used for the vectorization of maintenance work orders. A commercial tool called PoolParty Taxonomy & Thesaurus Management System (<https://www.poolparty.biz/>) has been used in accordance with the human-in-the-loop approach for creating and extending the

thesaurus. PoolParty offers a secure server backend, analyzes the material, and semi-automatically expands the taxonomies.

Humans have identified the essential terms in the maintenance work order text and categorized them under appropriate broader concepts such as functional failure, defect, and treatment actions shown in Figure 2. However, they have used NLP to perform structured data extraction through the process called **tagging**.

MD-Terex-Restructured (7)		Single word analysis	Multi word analysis
<ul style="list-style-type: none"> Artifact (3) <ul style="list-style-type: none"> Asset (18) Component (115) Functional Unit (29) Condition (2) <ul style="list-style-type: none"> Conforming Condition (2) <ul style="list-style-type: none"> Conforming Functional Condition (6) Conforming Physical Condition (2) Nonconforming Condition (2) <ul style="list-style-type: none"> Nonconforming Functional Condition (6) <ul style="list-style-type: none"> Artifact with Incomplete Function (1) Artifact with Missing Function (9) Artifact with Undesirable Behaviour (5) Incomplete Functioning (8) Missing Function (34) Undesirable Behavior (31) Nonconforming Physical Condition (4) Event (1) <ul style="list-style-type: none"> Failure Event (5) Function (1) Maintenance Treatment (8) Material Substance (5) Property (3) 		Word Annotation	
		Words	Classification
1	unit	I	
2	kit	I	
3	seal	I	
4	replaced	S	
5	installed	S	
6	pump	I	
7	new	X	
8	hydraulic	I	
9	160008	U	
10	completed	X	
11	sn	U	

Figure 2. Thesaurus and NLP for Hybridized Data Classification.

The next step was to create a knowledge graph which will also be used directly for root cause analysis since it captures the relationship among different maintenance artifacts. Hence, we first needed to build an axiomatic ontology based on OWL (Web Ontology Language). OWL would enhance the expressivity of the SKOS model since, without any internal relationships among the concepts of the thesaurus, it is just a dictionary for maintenance terms. The concepts of the SKOS model have been mapped in Protégé based OWL file in top-down and bottom-up approaches and Figure 3 is an

example representation of OWL ontology. In the top-down approach, top-level core classes have been defined, guided by the International Ontology Foundry (IOF). A bottom-up approach has been used to determine maintenance annotation in the thesaurus aligned with ontological classes (Ameri & Yoder, 2019).

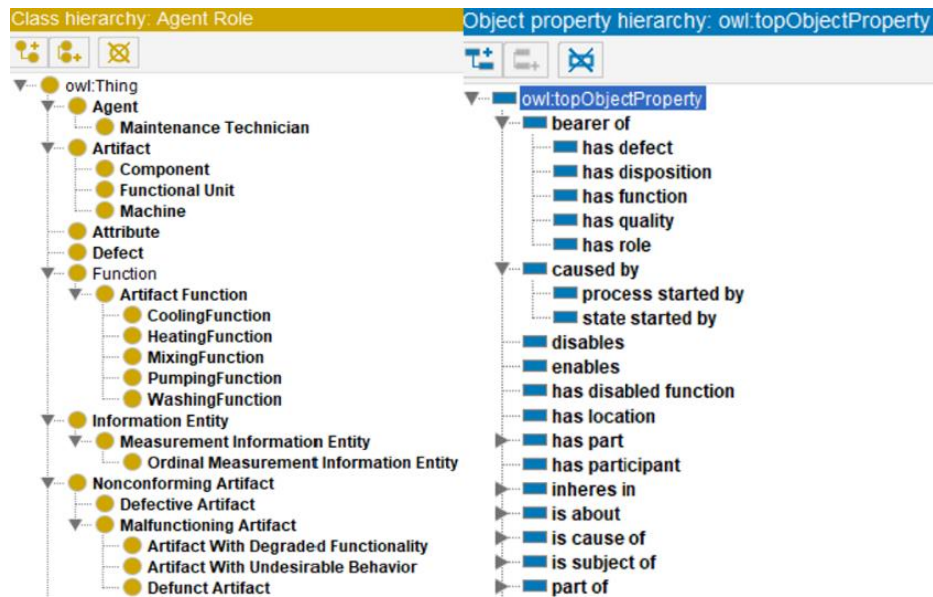


Figure 3. Owl Ontology Classes and Object Properties in Protégé.

Both the output of this OWL file and the SKOS thesaurus has been used as input to the java based tool to build triples for the work orders. OWL contains details of the appropriate linking properties along with the ontology classes to develop the triples, and the output of this java based tool is our desired knowledge graph. The graphs could be viewed in the web-based RDF (Resource Development Framework) Grapher to analyze potential failure root causes in the maintenance work order. An example of raw data is: “*Damaged air bracket cleaner, key switch not working,*” and Figure 4 illustrate how informative a knowledge graph looks compared to that single line raw data. It elaborates on information like air bracket cleaner, and switch are types of components, key switch is not working due to damaged air bracket cleaner. We can also understand the relationship

between each individual, and when analyzing larger datasets, these internal relationships result in extensive root cause analysis. Maybe this same damaged air bracket cleaner is the cause of problem in other operations. Rather than repeatedly going through the raw maintenance work order, we would build this knowledge graph to explore the entire history.

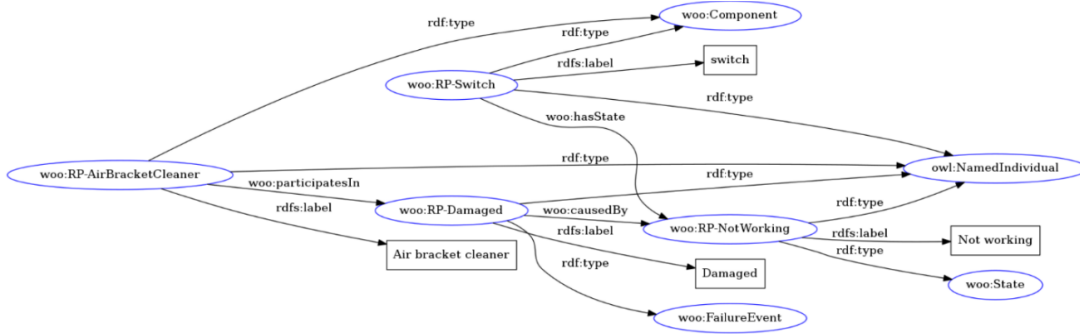


Figure 4. Visualization of Single Raw MWO.

We have used secondary data provided by NIST, which they collected from different manufacturing industries. Our data sets are descriptive since they consist of unplanned failures rather than any external manipulation. We could state that the failure data is collected by maintenance technicians' observation without any external intervention. We have used different tools to conduct our quantitative methodology, such as Nestor to apply NLP in thesaurus development, SKOS tool for thesaurus generation, Protégé for OWL ontology, and other Java-based tools to generate RDF triples and to conduct SPARQL query. Nestor uses NLP in the background to identify the repetitive terms, removing the unnecessary gaps and punctuation to perform structured data extraction from the raw MWOs with minimal annotation time-cost. SKOS tool is used for creating the top-level categories of terms and further populating the lower-level categories by the terms extracted from the raw data. Those top and lower-level categories

have been mapped in the Protégé tool. A java based tool has been developed to use the SKOS and Protégé tool output as input in the RDF and OWL format. Afterward, an online enterprise knowledge graph platform was utilized to conduct queries for finding necessary relationships in the database.

All of our methodology approaches are standardized, and it is possible to create a smart maintenance tool with the thesaurus-guided maintenance database development. Building the thesaurus is not hardcore data sorting, and hundreds of SKOS vocabularies exist on the web. Therefore, we believe that our proposed model can be linked and integrated with other vocabularies to enhance the semantic coverage of the knowledge graph. However, one shortcoming of the proposed methodology is since the thesaurus will be partially developed by humans, it will be time-consuming. They will also mitigate the effect as they can judge the quality of the data by decision-making and their past experience better than the NLP when data extraction is the preliminary task. Another shortcoming of the proposed approach is that the SKOS ignores the type of relations between two concepts and treats all relationships as the same. This caveat can be countered by superimposing more expressive ontologies on top of the lightweight thesaurus to enable more advanced reasoning.

Research Plan

The following specific tasks are planned to meet the objectives of the proposed project, followed by Table 1, which shows the timetable of our tasks.

Task 1: Creating and Extending Maintenance Thesaurus

The first step in developing the knowledge graph was to create a corpus of technical documents related to the maintenance domain, including MWOs, to be used as

the training dataset. Using techniques such as text mining and NLP, the relevant terms can be extracted and integrated with the taxonomy of SKOS concepts. The next step was creating the top-level categories of terms and further populating the lower-level categories with the terms extracted from the corpus. The thesaurus has three main concept schemes (collections), namely, Artifact, Maintenance Problem, and Maintenance Treatment. An alternate concept label could be created depending on different companies.

Task 2: Creating and Extending a Formal Ontology

An OWL ontology has been developed for the representation of the domain of industrial maintenance. The scope of the ontology will be determined based on the available datasets and motivating use cases. The final step was the creation of an OWL file in Protégé, and a modular approach will be followed for ontology development.

Task 3: Knowledge Generation and Expansion

The java tool is used to tag work orders one by one of a certain manufacturing company and develop the triples relationship. Later, all the relations were added to a master turtle file, and the RDF graph web service was used to visualize the knowledge graph.

Task 4: Validation of the Developed Semantic Models

This task entails applying the semantic models that have been built to support search, retrieval, and decision-making. The first step (4-1) is to define a set of KPIs using precision and recall performance metrics, and then the necessary computer programs will be developed to extract the values of KPIs from the raw data. Maintenance KPIs provide information about common problems, such as considering a problem where

we want to know the undesirable behavior of assets caused by defective artifacts. Now, our query returns ten records, and after comparing them with the text inputs, we found out that only six undesirable behavior is related to asset caused by defective artifact. So, the KPI for this problem will be what percentage of undesirable behavior of asset caused by defective artifact is truly positive. Similar to this, other KPIs could be what percentage of failure events identified by the query are actually the failure events modeled correctly in the knowledge graph, what percentage of the hydraulic leak from O-ring is actually the hydraulic leak from O-ring in the knowledge graph, and so on. Maintenance operations management decisions can be taken depending on these KPIs with certainty rather than time consuming in determining the solution. The second step (4-2) was to determine if the semantic knowledge models capture the semantic relationship between concepts. This was done by running necessary queries formulated in SPARQL and analyzing if the output data was showing the root causes. This is similar to putting semantic models to work and applying them to the maintenance diagnostics process.

Table 1. The Timetable of Tasks to Accomplish the Goal

Tasks	Year 1										Year 2							
	M 3	M 4	M 5	M 6	M 7	M 8	M 9	M 10	M 11	M 12	M 1	M 2	M 3	M 4	M 5	M 6	M 7	M 8
1																		
2																		
3																		
4-1																		
4-2																		
	Spring 21		Summer 21			Fall 21					Spring 22				Summer 22			

II. LITERATURE REVIEW AND SEMANTIC DEFINITIONS

According to Mohan (2015), The failure of the state-of-the-art NLP systems to generalize correctly suggests that they are unable to learn meaningfully from their training data. Text encountered in technical applications, like industrial processes, differs greatly from normal benchmarks in certain aspects, such as lexical, grammatical, and terminological variances, causing deployed NLP systems to perform poorly. Especially, maintenance text is frequently written in a style that resembles shorthand notation, with several stop words and no punctuations (Sexton et al., 2017). In addition, the reasoning behind the NLP analysis results is easily buried by incomprehensible computational black boxes obstructing human understanding. Dima et al. (2021) acknowledged the problem by suggesting an approach to Technical Language Processing (TLP) for the dataset explicitly containing industrial engineering case raw texts and mentioned ‘Nestor’ and ‘Redcoat’ as the ideal NLP-based TLP toolkits. Surprisingly, 'Nestor' was also employed in our research to extract structured data from the raw industrial maintenance data. So, it is easily derived from the approach these authors were trying to make, and we have implemented that in our paper. The authors then talked about the domain adaption among different sources which share similar syntactic structures and parts of speech (POS). Besides, the affordability of NLP due to its high computational cost and the concurrent TLP techniques such as the Convolutional neural networks (CNN), Support vector machine (SVM) to achieve similar NLP performance were addressed to make the engineering domain analysis simpler.

Gharehchopogh & Khalifelu (2011) elaborately compared Text Mining and NLP for identifying useful information from the raw text for all businesses. They did not use

any specific methodology to conduct their research but instead illustrated how to utilize both techniques depending on structured and unstructured data. They differentiated the ideas of data, text, and web mining and concluded that the model that uses both Text and Web Mining to retrieve structured data sets from the unstructured data is more successful than the obtained structured data. They also focused on the fact that Text Mining is used to find useful patterns in texts. In contrast, NLP deals with the underlying information/metadata and supplies text classification, categorization, document clustering, information extraction, summarization, etc. 1st step of the research tasks is clarified in this article: how Text Mining uses NLP to create the final Text Analytics model.

Ameri & Yoder (2019) discussed a similar methodology as Gharehchopogh & Khalifelu (2011) of using text analytics techniques to extract data from CMMS. However, they introduced a hybrid methodology by combining the human-assisted thesaurus development method to generate the formal knowledge graph. This knowledge graph uses the Simple Knowledge Organization System (SKOS) to show the semantic relationship between various entities in the maintenance domain. A Java-based tool is developed that uses the generated SKOS thesaurus as Resource Description Framework (RDF) format input, which results in a maintenance diagnosis output map. The researchers also mentioned that SKOS is widely accepted in companies, so the proposed MD thesaurus can be integrated with other vocabularies to enhance the semantic coverage of the knowledge graph. Our research plan is to implement this methodology in the manufacturing database we are working with and map the SKOS concept in the OWL ontology. The creation of an OWL file will allow us to overcome the limitation of

treating all relationships the same between two concepts the researchers' team mentioned in their work.

Gunay et al. (2018) also demonstrated text mining using two HVAC datasets to determine failure patterns, but their way of conducting the experiments differed from the previous two authors. They segregated the mining process into three steps: preprocessing datasets, clustering important terms (Ward's method), and identifying coexistence tendencies among the clustered terms. Their noticeable finding for dataset one was that HVAC maintenance has a much stronger relationship with the building type (research, admin buildings) rather than with building vintage (old or new buildings). R programming software was used to fix the punctuation (eliminating gaps, and capital to small letters). Figure 2 shows the conversion of the datasets into a mathematical form known as a document-term matrix. A Term frequency-inverse document frequency (TFIDF) score was applied to estimate the relevance of terms within a data set. The main difference between this research team and our methodology is that they used the association rule-mining (ARM) method to discover the relationship among terms. Three key concepts: support, confidence, and lift (Witten et al., 2016), were used to set aside frequent terms of the datasets. The association rule-mining was then performed on clusters that only contained a large number of interesting phrases. The authors developed the top 15 association rules based on their confidence and support, and FMEA analysis as well as box-whisker plots, were used to determine the failure modes.

Another research team, Brundage et al. (2021), very recently published an article where they summarized the processing steps of NLP working with the technical language for the clinical notes of an asset management system. There is no doubt that the raw

maintenance data is unstructured, and according to Ameri and Yoder (2019), human experts' intervention is needed to validate the generated models. Brundage et al. implemented this human-in-the-loop approach and named it Technical Language Processing (TLP) to tailor NLP tools to engineering data. The researchers described all the steps of the NLP process and provided a detailed description of how TLP Industrial leaders, standards organizations, professional societies, and researchers should work together in reality. It seemed from their work that TLP is comparatively a new methodology to work with, and the research community has started developing maintenance resources. Their research methodology directly intersects with our research tasks of using the hybrid approach to validate the model, and we will be able to help this research team accelerate the TLP development.

Sexton et al. (2017) also proposed a hybrid methodology in their research, showing a comprehensive description of raw maintenance log datafication. Datafication is a necessary step to transmit human contextual knowledge during the process of structuring the data in such a way that it creates value. They combined AI techniques for NLP, machine learning, and statistical processing augmented with human guidance to develop their modeling concept. They mainly focused on hybridized data tagging, where NLP is the way to optimize a human tagger's time investment. They concluded that tagging is the best way to address maintenance log data problems and introduced us to a new term called Support-Vector Machine (SVM) to increase the data precision. We think the gap between their and our research purposes is that we want to develop a formal ontology, whereas they focused on datafication. Also, we found it difficult to understand some of the technical methods they were trying to utilize in their industry case study

compared to the other papers we reviewed earlier.

Naïve Bayes (NB) and Support Vector Machine (SVMs) text classification approach is used by the researcher team Arif-Uz-Zaman et al. (2017) to extract accurate failure time data from two types of the dataset: work orders (WOs) and downtime data (DD) for the Australian electricity and sugar processing companies. WO data are often unreliable, refer to unplanned failure or defect due to machine breakdown, and DD contains machine stoppage time. None of the datasets referred to the root cause of the asset failure, and so the teams' main objective was to link those datasets to determine the reason behind machine downtime. The sequence of their data analysis was: labeling, text cleaning, constructing a keyword dictionary, text feature extraction (a bag of words), tokenization, and matrix build-up. SVM showed higher accuracy while they were extracting data from WOs, and only the SVM classifier was applied to the DD to label each as failure or nonfailure. Their resultant graph showed that both the DD and WO events appear to overestimate the failure rate, and the number of cumulative failures is equal to or less than the raw number of DD events. The researcher explained the classifying process in simple words, and we found many similarities, such as the use of SVM in our ongoing research.

Hodkiewicz et al. (2021) introduced us to a comparatively more recent term called prognostic health (PHM) technologies which can be implanted to detect potential failure. In order to make intelligent maintenance decisions about maintenance, it is necessary that PHM fits an organization and has enough past resources and pieces of information available. They focused on Structured Work and Corrective work since they account for approximately 80% of the maintenance work. Besides, a good explanation

has been found on whether to use preventive, predictive, or condition-based maintenance. They used two case studies to elaborate on measuring the effectiveness of a PHM initiative depending on different maintenance strategy output for three types of pump. They concluded that every corrective maintenance work order should have a detection code but concentrated on rethinking management matrices. We learned from this article that how PHM could be a game-changer in the future, and clearly, they agree with our research perspective that current maintenance matrices are not ready for fully machine-actionable knowledge as well as industry 4.0 world.

There are a few pools of other authors who worked with knowledge graphs and ontology together. Hossayni et al. (2020) are one of them who successfully developed a SemKoRe knowledge graph that gathers all failure data and shares it among connected users. The SemKoRe maintenance process includes diagnostics to determine the reasons for a failure and its impact and to apply the correct repair, which improves machine maintenance for failure occurrences. They mentioned two important drawbacks of traditional systems such as CMMS, and ERP, which included problems with sharing the maintenance data at two different locations and the lack of semantics in users, which worked as their research motivation. They developed a flat ontology using two types of machine parts, and when a failure occurs, the machines create instances of **Failure Occurrence Class** containing all the information about the failure, and hence, a knowledge graph is developed. The main difference between ours and this researcher's teams' work is that they are considering a different type of knowledge graph rather than SKOS, and they also categorized the deployment option (local, cloud-based, and hybrid) to protect business data. Besides, they started with a simple ontology model with only

two machines, whereas we are developing a detailed ontology base knowledge graph depending on real company data with a large number of machine parts.

Ringsquandl et al. (2017) created a RDF knowledge graph for Siemens smart factories by tailoring Ontology-Based Data Access (OBDA) for a smooth maintenance operation. They provided a graphical representation of how the digital twin works as an interface to the physical system allowing optimization and self-organization without interacting with the part. The major difference between our research is that they want to improve the performance of the system by identifying the missing information between the instances of the class. In contrast, we are developing the ontology-based knowledge graph primarily to determine failure. Their knowledge graph representation is also a bit different from ours since their knowledge graph talks about master, operational, and transactional data and enhances the vector space of log files generated by manufacturing equipment. After the OBDA was developed, a machine learning approach was used to identify the missing entity in the RDF triples (entity, predicate, entity). They mentioned that this missing data often is a result of new or replacement machines.

Categorizing the maintenance failure with the CMMS drop down menu is a common process many companies do to mimic human interpretation to analyze maintenance logs. However, collecting data is unique to AI, and one of the possible approaches to get a similar human conducted output is to hybridize the data collecting and sorting process. Sexton et al. (2017) conducted a study on manufacturing logs to determine how NLP as a part of AI can be used effectively to extract useful information as human participation can obtain. Their approach was to solve the problem through hybridization, datafication of the logs, and statistically analyzing the data trend. The

authors did not have any controlled vocabulary like us, rather, they only used NLP for tagging to assign characteristics to the data instances simply. For our, we did two step tagging, starting with NLP and then building a controlled vocabulary since our goal is to show the triples by graphical representation. To measure the quality of the automated tagging, texts with no unknown tags were considered fully datafied. But for texts with no known tags, a linear-kernel support-vector machine (SVM) was used to pick up the tag patterns. They also talked about diagnostics depending on the occurrence rate for certain tag combinations and future work needed for taxonomy development, which means what we are doing in our research.

Without agricultural improvement, humankind cannot exist, and sadly, no good designed knowledge graph is available for the cultivation sector. To fulfill this need, Qiao et al. (2017) designed a knowledge graph consisting of schema and data layers from the agricultural thesaurus. The graphs are shown using the Echart tool. The schema and data layers combine to generate a huge graph known as a knowledge graph. They started with the schema level, which allows them to readily distinguish between concepts (classes) and entities (individuals). They defined relationships between individuals and classes in the data layer. The knowledge graph is then stored as RDF triples (entity, relation, entity) in the graph database by using Jena. The biggest similarity between their and our work is that the paper developed their agricultural knowledge graph as we did for our maintenance knowledge graph, and it is the foundation for the semantic-based knowledge graph. However, their focus was on building a knowledge graph to show the relationship among the entities and concepts (e.g., hybrid rice and paddy rice). Whereas we modeled the manufacturing problem statement collected by the operators to identify the root cause

of the failure.

McKenzie et al. (2010) used The Natural Language Toolkit (NLTK) to detect faults from the Condition-based maintenance (CBM) database containing 100,000 individual vibration data and historical maintenance records from Army helicopters. This research team realized long before Dima et al. (2021) that the existing systems need to be tailored to classify the unique dataset for future research. They used the inspection description, which accumulates forty percent of the entire record as their input data, which is quite similar to using problem statement description in our research. However, they conducted a partial parsing approach, also known as **Chunking**, to only detect the necessary information like inspection info, date, and time. In contrast, we did full text analysis so that no failure was left out. Also, the NLTK toolkit is open source, and the python programming language is well written, ensuring easy manipulation depends on specific needs rather than domain restrictions. On the contrary, we have used domain-specific text mining to fulfill our goal of making a manufacturing maintenance ontology. Furthermore, their Part-of-Speech (POS) tagging had a default tag NM for all the untagged word in their trained tagger system, which was the researchers' source of error when our tagging strongly required human intervention to achieve utmost precision. Regardless of how they did the initial data processing, they formed triples to show the relationship among the three extracted parts (inspection info, date, and time) after chunking. The researchers also analyzed the performance of the POS tagging and chunking using standard matrices, which we would do during our validation process.

Vibration data, engine oil debris measurements, and other indications are used in Condition-Based Maintenance (CBM) to determine maintenance schedules and

procedures. Hence, Bokinsky et al. (2013) concentrated on modifying the components of the Natural Language Processing tool for detecting CBM-related status. They used the same NLTK toolkit as McKenzie et al. (2010), and their work was kind of an extension to the helicopter maintenance data but for another V-22 Osprey project management database. Unlike us, their records are stored in Maintenance Action Form (MAF) records and the NLTK has been used to extract necessary information. The pre-processing of data is pretty much the same, including tokenizing, sterilizing, and POS tagging using the N-gram algorithm. The researchers used four taggers for extra precision, and if the first tagger could tag the words, it would stop there; otherwise, the word would be automatically sent to the next tagger. Chunking has been used to represent the noun, verb, and reference format meaningfully. Lastly, a file of hand-tagged and hand-chunked data was created for evaluation purposes; the tagger had 96.59% accuracy.

Gao et al. (2020) proposed a text processing pipeline using technical language processing for unstructured data during the need for corrective maintenance work. This ensures knowledge about the failures as well as differentiates between the need for replacement and repair. Although they preprocessed the data like McKenzie et al. (2010), they proposed Named Entity Recognition and classification (NERC) to identify the action-state-item from the unstructured data. N-gram named dictionary generation for the maintenance state was the following stage, partially similar to our maintenance thesaurus. The steps of the pipeline were illustrated well with detailed picturization and description, and validation of the pipeline was done afterward using part-of-speech (POS) tagging as a baseline which is also similar to the helicopter database validation. The POS tagging is described elaborately, which will guide us during our model validation.

The extent of maintenance data and how it is collected differ per industry. Many failed attempts to structure data by enforcing controlled vocabulary and problem code assignments for MWOs have been made in the past. To determine Median Time against Fail/MTTF, Sexton et al. (2018) contrasted a data-driven tagging method to a rules-based expert system (represented by Kaplan-Meier estimation and Weibull distribution models), which has indeed guided us to select the appropriate methodology for our own maintenance data extraction. In rule-based data processing, thorough human intervention is needed to transform the unstructured WOs into a predetermined format using explicit rule sets comprised of conditions between one to three logic statements. In comparison, NLP is employed to construct a machine learning pipeline to further capture the correct words in the WOs in data-driven tags. However, only using the NLP can fail to identify the important terms necessary to build the semantic relationship. As a result, they enforced future work requiring NLP automated data extraction with human-in-loop, which is the basis of our model development.

Similar to the manufacturing industry, the volume of biomedical literature is increasing, and Spasic et al. (2005) proposed a text mining pipeline to further create a conceptual biomedical ontology framework. The researchers provided a very clear elaboration on terminology, which is the link between the text and ontology, as well as the problems while linking them, such as term ambiguity (Promoter means binding site in a DNA chain, but in chemistry, it increases catalyst's activity) and variation (Advil, Brufen, Nurofen all refer to ibuprofen). Their proposed text mining methodology is very similar to the previous researchers, consisting of tokenization, part-of-speech tagging, stemming, and lemmatization. In our thesaurus, we implemented lemmatization by

putting the word to its base form (e.g., broken is an alternate for the break, and leaking and leaked are alternates for the leak). Likewise, McKenzie et al. (2010) and our thesaurus, named entity recognition (NER), have been proposed to extract and store the complex biomedical information to link with the ontology. Besides, a comparison between passive ontology and ontology driven information extraction (IE) was discussed. The authors preferred ontology driven IE due to the fact that passive ontology has a tendency to link between text terms and concepts of ontology without any explicit relation. On the other hand, ontology driven IE analyzes the constraints carefully, which we have also implemented on top of the thesaurus for semantic representation.

The asset intensive industries deal with a lot of maintenance data as a part of the operating cost and sudden failure can cause monetary and organizational data safety consequences. Rather than using external industry data for failure analysis, it is possible to analyze and cleanse the large internal company CMMS data in a timely manner by using rule based approach with a conflict resolution step. When two rules link differing text to the same field, conflicts are identified and appropriately noted, and the conflicts are resolved by modifying as needed. Hodkiewicz and Wei Ho (2016) analyzed rule based reliability of five mining sector data which requires lifetime data distribution to distinguish between different failures and corrective/preventive measures. Different WO issues, such as: not recording the utilization data of assets, inaccurate cost data for replacement/missing parts, alignment of subunits (turbochargers are commonly replaced when the engine is replaced), and WO duplication, were identified in the source data. The researchers developed a rule based syntactic data cleansing tool **DEST**, with an **if condition platform**, which relied on occurrences of words, whereas we did a semantic

analysis to link concepts. Although semantic analysis is unscalable, developing 407 rules for only a single case study, such as the researchers did, is very time consuming and lengthy. Besides, the cleansing tool is more like our thesaurus where text is only classified but we have implemented OWL ontology to give semantic meaning to the texts.

To this date, there are currently no industry-wide guidelines for recording and evaluating unstructured data sources within an industrial site. Navinchandran et al. (2020) utilized NLP analytics to extract concepts from Maintenance WO and measure their effect on key factors (cost, time) to determine good or bad behaviors to achieve an optimized maintenance strategy rather than further development of a failure detection model. In simple words, if a MWO talks about the dollar amount spent on assets, parts, or any maintenance related cost, the decision maker could further break down the analysis to determine which assets have the strongest relation with cost. To do that, text preprocessing is needed to rectify any numeric entries, such as for time length; the lower bound should be a minimum of 5 minutes. Like us, the researchers have used Nestor to categorize the single and multi phrase words that carry important information regarding solutions, items, and so on. These text inputs, as well as the KPI inputs, could be used as an input of explanatory models, such as Continuous regression models, probabilistic models, and classifiers, to measure the effect on the output. The researchers later developed a decision tree and corresponding Gini importance to put weight on various assets linked with the KPI. Although we have used a similar tool Nestor, for initial processing, our goal is to build a smart maintenance tool, whereas the research team wanted to investigate the MWO relation with the performance indicator.

Extracting critical information has been a long vision of the owners of expensive manufacturing equipment to reduce downtime and determine an optimal maintenance schedule. Devaney and Ram (2005) made an interdisciplinary approach consisting of an advanced text analytics algorithm, artificial intelligence (AI), and OWL framework to address this issue. More specifically, their objective was to identify the component categories (e.g., Clamps), identify problem categories (e.g., hydraulic oil leak), and finally learn the distribution of the problem (e.g., hydraulic oil leak accounts for 20% of clamp problems). To satisfy the goals, the authors have used OWL to classify the input data under categories, and the categorized output is used to create a case library. The patterns in this output could predict future failure and diagnosis. The authors further proposed bootstrapping clustering algorithm to give natural categories for business process analysis. All of these steps ensure the proper utilization of the Case-based Reasoning Engine (CBR) as a next step which will provide recommendations for each hypothesis generated in the engine, which would aid the experts in making the right decision. Although this research team has taken similar steps to us, our goal is to analyze graphically presented triples relationship related to failure from a vast dataset. Whereas this paper focuses on combining three different models to have a solution to the detected failure.

Our learning from this whole literature review is that the industries are not ready yet for fully automated maintenance since the raw data collection comes from both humans and machines. We need to create a semantic system that will show the relationship between the components and grow over time. By growing, we meant that we would have humans in the loop to feed the AI system continuously, and the AI would

eventually generate the cause and solution of the failure. As a result, our research will start with thesaurus development to give us a controlled vocabulary and, eventually, a knowledge graph. An OWL ontology will be developed to represent the industry domains depending on the knowledge graph output.

Linguistic and Grammatical Maintenance Text Analysis

As mentioned earlier in Chapter 1, NIST provided us with a few companies' manufacturing raw maintenance data. For the sake of confidentiality, we will be using the name **Test Maintenance Company** instead of the actual company name for our whole research methodology. The general rule is that whenever a breakdown happens with any machine or its parts in any manufacturing company, the maintenance technician would list them out as problem statements along with their resolution. This thesis only focuses on the problem statement to determine the root cause of the problems first, and then, the solutions can be added as future work. In this section, we will conduct a structural analysis of some work orders, showing why having a knowledge graph is better than just the raw maintenance texts. Table 2 represents a few Test Maintenance Company work orders which have been used for our knowledge graph development, along with the questions we asked during our model development. We can clearly see the inconsistency throughout the examples, such as operators having used both upper case and lower case. Words like GEN, and HYD are incomplete, and the model developer had to assume that **GEN** means **generator** and **HYD** means **hydraulic**. Besides, in any of these problem statements, the specific machine or part name and number are missing. It is likely that the company has multiple machines and parts, but it is impossible to determine which specific machine or part is having the issue from any examples. For example, one could

ask which fuel float pump is bad, as there could be multiple. Also, what does it mean by bad? Does it mean that there is a leak, or is it just not operating?

Table 2. Test Maintenance Company Example Problem Statements

Num	Problem Statements	Example questions could be asked by the model developer
1.	GEN NOT WORKING, KEN W/ GENIE ADVISED TO CHANGE RESISTOR AND REPLACE POT SWITCH INSIDE BOX.	What is GEN?
2.	fuel pump float is bad stuck doesn't work sending unit is bad	Fuel Fuel pump float of what machine? What does it mean by bad?
3.	code 23/no functions, faulty lower module	What is code 23/? Is it different than no functions? What is the location of a faulty lower module?
4.	Unit does not move or work, tech found the gcon shorted out	What are Unit and gcon?
5.	Charger Defect	Defected charger caused what issue? The charger of which machine or component?
6.	control box issue box board is faulty	upper control Control box of what machine?
7.	Diagnos with genie for 154 egr code /clean connections at sensors /test and clear codes /	What is genie and what is the problem here?
8.	OIL LEAK.	Oil is leaking from where?
9.	Breakage	Which part or machine is broken?
10.	HYD LEAK BLOWN ORING AT CENTER POST ROTATOR	FOUND What is a HYD leak?

Technical jargon like **code 23**, **154 egr code**, **gcon** has been avoided during the thesaurus development since they do not add any value, and only the maintenance people understand the best meaning of these. In addition, some of the problem statements, such as problems 5,8, and 9, are incomplete, and there is no further information regarding the

problem. We are assuming there could be multiple machines that need a charger, and the machine number is missing. Only breakage and oil leak do not tell anything about the trouble making machine. There are unnecessary spaces between the words, and sometimes missing punctuation makes it challenging to understand the meaning of the sentence. In problem statement 4, one can see the word **unit**, and it took us quite sometime to understand that by **unit**, the operators mean a machine, but not a group of people or other entities the term ‘unit’ refers to.

It is difficult for domain experts trying to find the root cause of the maintenance problems only by going through the raw data in CMMS. Besides, it becomes time-consuming, and sometimes the primary source of the breakdown could not be found due to the technical jargon, missing/incomplete sentences, let alone understanding the relationship among the individual machines and parts and their associated failures. Hence, our proposed knowledge graph will come in handy as we will only identify the keywords in the maintenance work order and cluster them under classes in our maintenance thesaurus. We have eliminated all the unnecessary stop words and spaces and gradually built semantic relationships among them, and the representation is the knowledge graph. The knowledge graph will unwrap how one maintenance failure is causing another failure as well as the nature of the root cause. Figure 5 clearly illustrates how a knowledge graph identifies the root of the problem rather than the tip. The maintenance work order describes both the observation and diagnosis. In the upper side of the figure, the present and past observations can be seen. These observations are of different types, and without any diagnosis on the historical data available, it is hard to understand what has actually happened and how those events are causing more

breakdowns. Such as, in the lower side of the example figure, defective artifacts and artifact malfunctioning are observed. However, the deeper diagnosis is that other defects cause the malfunctioning of an artifact. Hence, we want to know the root of the problem rather than the tip, and in this process, we create a knowledge graph that shows the internal relations among individuals that are not understandable only by reading the raw text.

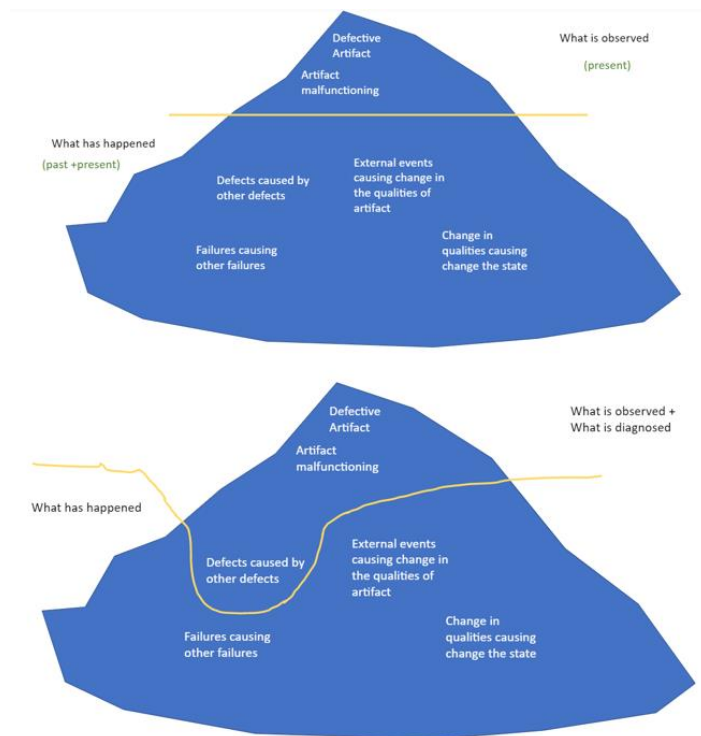


Figure 5. What is Observed vs. What is Diagnosed.

Semantic Technology Definitions

The World Wide Web Consortium (W3C) is an international organization that creates open standards to ensure the Web's long-term growth. The Semantic Web, often known as Web 3.0, is an extension of the World Wide Web based on World Wide Web Consortium specifications (W3C). The Semantic Web's purpose is to create a common foundation for data sharing and reuse across applications and make Internet data

machine-readable (*Semantic Web* - W3C, n.d.). We will use SKOS, RDF, RDFS, and OWL standards to implement our methodology. These standards allow data from various sources to be linked and integrated, which is required by the tools and software we will be using. So, we will introduce the basics of these four W3C technology standards by briefly describing each standard in the next sections.

Simple Knowledge Organization System (SKOS)

People have been using Knowledge organization systems to organize large collections of objects such as books or museum artifacts. Knowledge organization systems (KOS), and more specifically, controlled, structured vocabularies, are integral parts of data classification systems (SKOS Simple Knowledge Organization System - Home Page, n.d.). However, these controlled vocabularies need to be structured in a way that both humans and machines can understand the meaning. Linked open data and linked open vocabularies are Semantic Web technologies that allow for the publication of controlled vocabularies on the web in a way that both people and machines can understand. Besides, a controlled vocabulary allows certain content or knowledge to be organized so that it may be conveniently recalled at a later time. Here comes the Simple Knowledge Organization System (SKOS), which provides a standard way to represent the KOS or controlled vocabulary as machine-readable data. It is a data-sharing standard that integrates several domains of knowledge, technology, and practice.

The Simple Knowledge Organization System (SKOS) provides better organizing of the vast amounts of unstructured (i.e., human-readable) information on the Web, providing new routes to discovering and sharing that information (SKOS Simple Knowledge Organization System Primer, n.d.). It was built on several pre-existing

Semantic Web formal logic and structure standards. One of them was Resource Description Framework (RDF) which will be discussed as the later semantic technology. RDF provides a common data abstraction and syntax for the web, and since SKOS is based on RDF, its data is expressed as RDF triples. Within the Semantic Web framework, SKOS is a data model for different KOS: thesauri, classification schemes, subject heading systems, and taxonomies. Furthermore, SKOS is a data sharing standard and so allows low-cost transferring of the existing KOS to RDF. Now we know a constructive definition of the SKOS data model, but one might wonder at this stage what how does the SKOS model look like, and Figure 6 will pour some light on that. Machines or components that are leaking oil are considered to participate in **Undesirable Behavior** since it is not expected from them. However, the difficulty is that the workorders consist of many types of leaking problems such as leaking oil, leaking fluid, hyd leak, lube leak and so on. Hence, in the SKOS model, **Leaking** is the **broader concept** for all types of leaking (in this example figure: Leaking Oil), and **Undesirable Behavior** is the **broader concept** of Leaking. The good thing is that SKOS provides scopes for creating an alternate label, so leaking fluid, hyd leak, lube leak etc., could be listed as an **alternate label** to Leaking Oil. Furthermore, users can also use **related label** just in case they want to analyze related individuals. In this example, an Oil Spill is related to leaking oil, so a related label has been shown. This explained example is for only one individual in the SKOS model, and we have defined other necessary individuals to construct the SKOS thesaurus model.

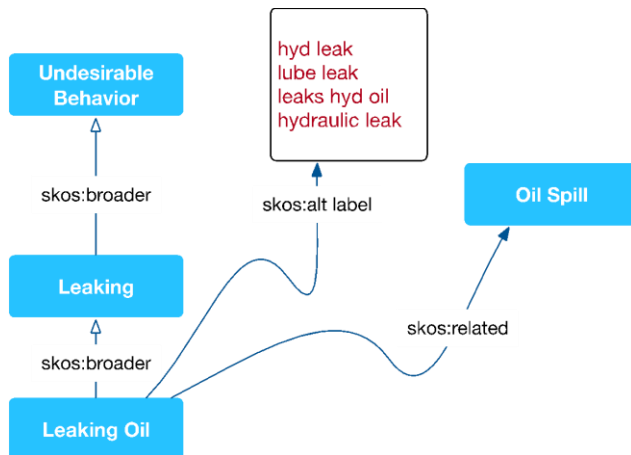


Figure 6. SKOS Model Example for Leaking.

Our ultimate goal is to build a knowledge graph based on the SKOS model that can combine and connect heterogeneous data on a semantic level. Such as, we know the broader/narrower concept of leaking, and the alternate/related labels of leaking, but what if we want to know the entire history from where the leaking is coming, why the leaking is happening, and what the leaking caused to other parts? These questions would be answered by another model, which would be described eventually by OWL ontology models. Since SKOS models lack the expressivity of heavyweight, axiomatic ontologies like OWL models are considered lightweight ontologies. However, SKOS models can be created reasonably quickly without the need to spend a lot of money creating complex, logic-based ontologies for many applications that only need fundamental semantics in terms of the structural and lexical links between different things.

Resource Description Framework (RDF)

RDF, or Resource Description Framework, is a W3C standard model for data interchange that is used for representing highly interconnected data (RDF - Semantic Web Standards, n.d.). RDF is the foundation of the Semantic Web, and all data in the Semantic Web is represented in RDF, including schema describing RDF data. While

there are many conventional tools for dealing with data and, more specifically, for dealing with the relationships between data, RDF is the easiest, most powerful, and most expressive standard. There have been various syntaxes to write it down. The original was called [RDF/XML](#); XML was used because it was standardized and flexible and also because one of the original RDF use cases was to add arbitrary metadata to web pages—the idea was that an additional block of XML would fit well into an HTML file’s head element. As it turned out, using XML to represent arbitrary collections of relationships could get verbose and messy. Now, most people use Turtle, which is much simpler and a W3C standard. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

RDF Triples Structure

RDF describes the data in a three-part structure statement, namely triples, consisting of **resources** referred to as the subject, predicate, and object (“Learn RDF,” n.d.). The subject is the entity identifier, the predicate is the attribute name, and the object is the attribute value (What Is RDF?, 2021). The subject, predicate, and object are actually represented using URIs (Uniform Resource Identifiers). Now, if we consider the 2nd workorder in Table 3 from our Test data, “*THE UNIT NOT STARTING. FOUND A DEAD BATTERY,*” we can create two RDF triples statements from here. They are: Unit **has state** not working and not working **is caused** by a dead battery. The URIs for the subject, predicate, and object are:

unit-> <http://infooneer.txstate.edu/ontology/MWOO/WO2-Unit>

dead battery-> <http://infooneer.txstate.edu/ontology/MWOO/WO2-DeadBattery>

not working-> <http://infooneer.txstate.edu/ontology/MWOO/WO2-NotStarting>

caused by-> <http://infooneer.txstate.edu/ontology/MWOO/causedBy>

has state-> <http://infooneer.txstate.edu/ontology/MWOO/causedBy>

Table 3. Test Data Example with RDF Triple Structure

Problem Statement WO2	THE UNIT NOT STARTING. FOUND A DEAD BATTERY		
Triple Statement Structure	Subject (Entity identifier)	Predicate (Attribute name)	Object (Attribute value)
Triple 1	unit	has state	not working
Triple 2	not working	caused by	dead battery

If we look at the URIs of **unit**, **dead battery**, and **not working**, they all belong to the second work order or WO2, which allows an absolutely clear understanding of what we are talking about. Besides, URIs are consistent across databases and enable us to create linkages between subject and predicate. This way, the same resource can be the object of some triples and the subject of others, which lets us connect triples into networks of data called the **RDF Graph**, which will be talked about in the next paragraph. If we carefully look at Table 3 above, we can see how the **object** in Triple 1 became a **subject** in triple 2.

RDF Graph

The RDF graph is a bunch of nodes connected to each other by edges where both the nodes and edges have labels. Figure 7 visualizes the RDF graph of the above mentioned example. In the top side of Figure 7, the resources are labeled, such as **Unit** is a **machine**, **not working** is a **defunct state** and **dead battery** is a **defective artifact**. These labels are the normal concept in the SKOS data model mapped as ontology classes in the OWL ontology. To add more relations to the graph, we simply need to add more

triples rather than making any structural change to the database. Now if the problem statement was “*THE UNIT NOT WORKING AND LEAKING FLUID. FOUND A DEAD BATTERY AND HYD TANK WAS LEAKING*,” we can add additional two triples on the knowledge graph, and they are the following: Unit participates in leaking fluid (**Triple 3**) and Hyd tank participates in leaking (**Triple 4**). The below side of the Figure 7 is a representation of how the RDF network looks like with additional triples. This is how the RDF model triples the power of any given data piece by giving it the means to enter endless relationships with other data pieces and become the building block of greater, more flexible, and richly interconnected data structures.



Figure 7. Test Data Example with RDF Graph.

RDFS, or RDF Schema, is W3C standard specialized language for describing RDF vocabularies and data models. It is lightweight and very easy to get started with. In fact, many of the most popular RDF vocabularies are written in basic RDFS. The goal of RDFS is to allow data created in different semantic technology to be connected via RDFS

("Learn OWL and RDFS," n.d.). The use of RDF does not require any schemas.

However, the commercial and open-source tools that can understand the RDFS vocabulary make it easier for applications to build user interfaces around RDF-based applications, integrate data from disparate datasets, and more. Whereas RDF is a graph database, RDFS is fundamentally about describing classes of objects. The `rdfs:label` property provides a human-readable name for the resource being described. This is especially helpful for reports and applications that use this data.

Now we will look at couple of examples from our Test Maintenance Company dataset where RDF is the object class, and RDFS is fundamentally about describing object classes. It is visible in Figure 8 that the schema or the structure of our example is in triple format, which we are utilizing for conducting the SPARQL query in the validation stage (What Is RDFS?, 2021). A SPARQL query can be executed on any database that contains RDF triples. The OWL ontology provides the semantics of the RDF dataset. Using triples made up of subject, predicate, and object, SPARQL interprets the data as a directed, labeled graph. Consequently, a SPARQL query is made up of a number of triple patterns where the subject, predicate, and object are all variables. The variables' answers are then retrieved by comparing the query's patterns to the dataset's triples. Going back to our examples, **rdf:type** predicate means is an instance of the following class. So, the first two lines in the 1st query at the upper side of Figure 8 are being used to declare **comp** (subject) as a type of **Component** class (object) and **def** (subject) as a type of **Defect** class. On the other hand, in the 2nd query at the lower side of Figure 8, **rdfs:label** is used to refer to a specific instance **not starting**. **Not Starting** is a type of **State** class, where **State** consists of several instances such as not moving, not

functioning, not working and, so on. The 1st query retrieves the components that are the bearer of defects, and the 2nd query retrieves states caused by other class instances.

```
SPARQL query:
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX woo: <http://infoneer.txstate.edu/ontology/MWOO/>
SELECT ?comp?bf?def
WHERE
{
  ?comp rdf:type woo:Component .
  ?def rdf:type woo:Defect .
  ?comp woo:bearerOf ?bf }

SPARQL query:
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX woo: <http://infoneer.txstate.edu/ontology/MWOO/>
SELECT ?st ?ca
WHERE {?st rdfs:label 'not starting'.
      ?st woo:causedBy ?ca}
```

Figure 8. Use of RDF-type and RDF-label in SPARQL.

OWL Ontology

As described earlier under **SKOS**, the Test Maintenance Company thesaurus is a dictionary of controlled maintenance vocabularies which are classified under several concepts' schemas. However, to create the knowledge graph, we need more than just concepts; we need to link the concepts. Here comes the Ontology part, which describes the concepts and the relationship between them in a domain. Ontology ensures a common understanding of information and represents complex knowledge about the concepts (OWL Web Ontology Language Overview, n.d.). Different ontology languages provide different facilities. OWL, a standard ontology language from the World Wide Web (W3C) Consortium, is the most recent advancement in this field. OWL is a computational logic-based language such that knowledge expressed in OWL can be exploited by

applications instead of just presenting information to humans (“Owl 101,” n.d.). OWL provides more vocabulary together with a formal semantical structure, enabling greater machine readability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S). Formal ontologies offer a context or meaning that both humans and machines precisely understand.

Foundational concepts in ontologies that are domain independent and can be used across domains can be reused, thanks to ontologies. We will be using Protégé to develop our Owl Ontology, as OWL makes it possible for concepts to be defined as well as described. Protege is an open-source tool that allows developers to create and manage terminologies and ontologies. Protégé also permits the use of a reasoner that can identify which concepts fall under which definitions and determine whether all of the statements and definitions in the ontology are mutually consistent. Therefore, the reasoner can aid in maintaining the hierarchy appropriately.

III. METHODOLOGY FRAMEWORK

In this chapter, we will describe how our methodology has been phased in and tied up together with semantic technologies. We will briefly describe how each tool has been developed, starting from the raw data collection to validation, which satisfies each task introduced in Chapter 1 and eventually resulted in our proposed methodology.

Figure 9 outlines the steps needed to complete each task as well as the highlights of each step are bulleted along with their formal model. Task 1 included building the thesaurus from the raw data of Test Maintenance Company. The maintenance work order (MWO) in excel format was the output of raw data collection and input of the thesaurus using the SKOS tool. The 2nd task was to create the OWL ontology and map the concepts of thesaurus in the ontology. Protégé tool was used to develop the OWL ontology, and it contains the annotations for the subjects, predicates, and objects, which were obligatory to develop the knowledge graphs in the next task. The OWL formatted output from Protégé, as well as the SKOS formatted output from the SKOS tool, has been used to generate the knowledge graphs in Task 3. The ultimate goal of generating these RDF knowledge graphs (serialized in in turtle format) is to facilitate root cause analysis of breakdowns and prospective failures. Finally, we used the RDF knowledge graphs (turtle files) to conduct queries and developed logic to express additional meanings that could be inferred from the dataset.

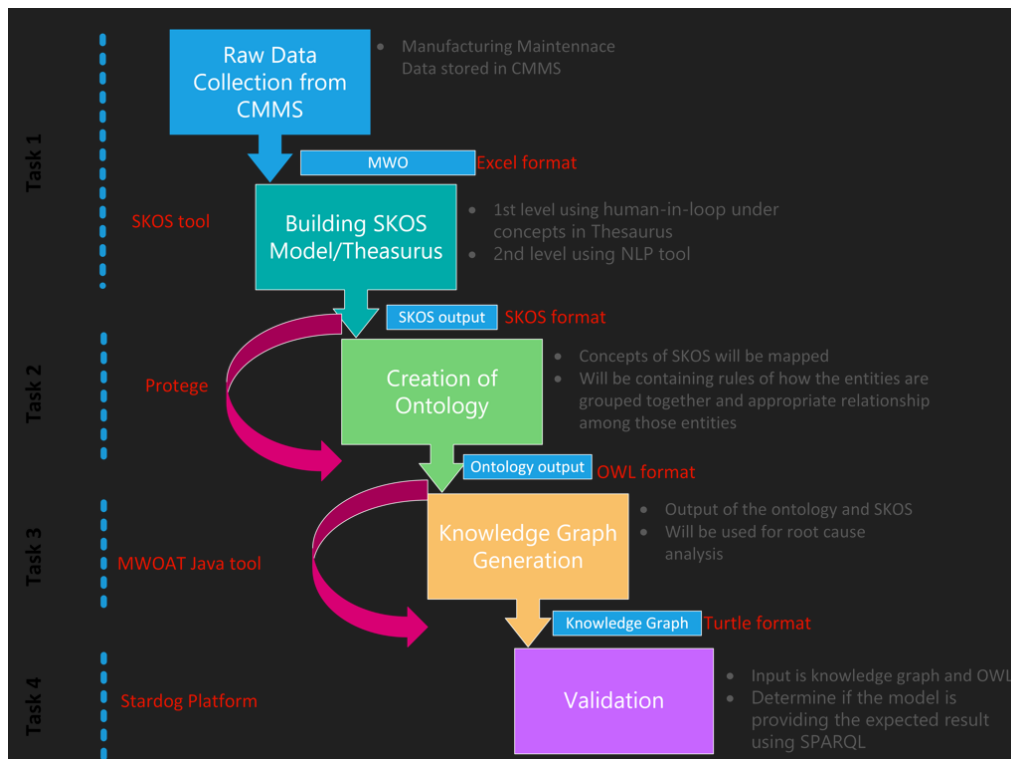


Figure 9. Methodology Framework from Start-to-End.

Task 1: Thesaurus Development

SKOS model has been used to develop the thesaurus-the first step in our knowledge graph development. A tool called INFONEER SKOS Tool or SKOS Tool, for short, has been developed for the generation and expansion of SKOS models (Ameri et al., 2020). SKOS tool provides a floor for easy classification of data by providing means for tokenizing and annotating documents using SKOS concepts, in our case, the maintenance data. The SKOS Tool runs as a Django web application. Python is used by the free and open-source web framework Django to implement the model-template-view paradigm. Numerous additional libraries, including BeautifulSoup4, are included in a virtual environment along with Django to support the tool's operations. Python was used to create the application's back end, whereas HTML and JavaScript were used to create the front end. At Texas State University, the most recent stable release of the web

application is installed on a development virtual machine running Red Hat Enterprise Linux, making it accessible to a select group of users via Secure Shell (SSH). Below are elaborately described the elements of our maintenance SKOS tool.

SKOS Elements

Concepts. The knowledge organization system is viewed as a **Concept Scheme**, including a set of **Concepts** in the SKOS data model or SKOS tool. URIs identify these SKOS concept schemes and SKOS concepts, allowing anybody to refer to them unambiguously out of any context and making them part of the World Wide Web. URIs are similar to URLs (Uniform Resource Locators), and often look like them, but they're not locators or addresses; they're just identifiers. The concepts are the primary building block of the SKOS data model, and the SKOS tool is required for initializing concept organization in our maintenance thesaurus.

This SKOS tool consists of certain classifications by grouping into **concepts** to enable easy identification of the vocabulary. Concept grouping will allow the maintenance terms of the retrieved Test Maintenance Company data problem statement to fall under the right bucket of classification. Semantic relation properties of the SKOS tool allow SKOS concepts to be related to one another and support hierarchical and associative relationships among SKOS concepts. For our maintenance data, we have different **concept schemes, top concept and normal concept** having hierarchical relations among them, which we have seen in Figure 2 previously. A good example of the **top concept** is Artifact along with a narrower classification into the normal concept. Also, in multiple concept schemes, our current maintenance SKOS concepts can be mapped to other SKOS concepts if necessary.

Labels. The labels are the descriptor of the concepts, and the concepts could be labeled with lexical strings. The labels are of three types: preferred (skos:prefLabel), alternate (skos:altLabel), and hidden label. When generating or developing human-readable representations of a knowledge organization system, the preferred and alternative labels are useful due to their ability to give authorized and unauthorized name to a concept. The hidden labels are valuable when a user interacts with a knowledge organization system via a text-based search function. We are only using preferred and alternative labels while generating our maintenance SKOS data model. Each concept in SKOS has exactly one preferred label (skos:prefLabel) and can have multiple alternative labels (skos:altLabel). Hence, the community users can add more labels to the shared open source data which results into thesauri enrichment and validation. Description can also be provided for each SKOS concept in plain English to comprehend the meaning. A good example from our maintenance SKOS model in Figure 10 is: **hyd leak** is an alternate label for the preferred label **hydraulic leak** under **Undesirable Behavior** in the **Condition** Concept Scheme. Depending on the need, one can add the scope note, narrower concept, related label, and hidden label for the concepts.

Edit concept

URI: <https://infoneer.poolparty.biz/MaintenanceDiagnosisThesaurus/803>

Preferred label:

hydraulic leak

Definition:

A leak from a hydraulic machine

Scope notes:

Created: 2022-04-25T20:36:52.481Z
Last modified: 2022-04-25T20:37:15.838Z

Alternative labels:

hyd leak

Hidden labels:

Broader concepts:

--

leaking

Top concept of concept schemes:

--

Narrower concepts:

--

Related concepts:

--

Figure 10. Partial View of User Interface in the Test Maintenance Thesaurus.

Semantic relationships. Semantic relations in the SKOS data model are the links between SKOS concepts. We generated this hierarchical semantic relationship referred to as the **broader concept** (skos:broader) and **narrower concept** (skos:narrower) to link the top concepts with the normal concepts. This is particularly useful while using the java tool in the knowledge graph development phase, which can be seen in Figure 11 below. The java tool not only shows the relation between the thesaurus concepts but also the appropriate OWL ontology class for each concept. **Functional Unit** is the **broader concept** for **motor**, and inversely, the motor is a **narrower concept** for Functional Unit.

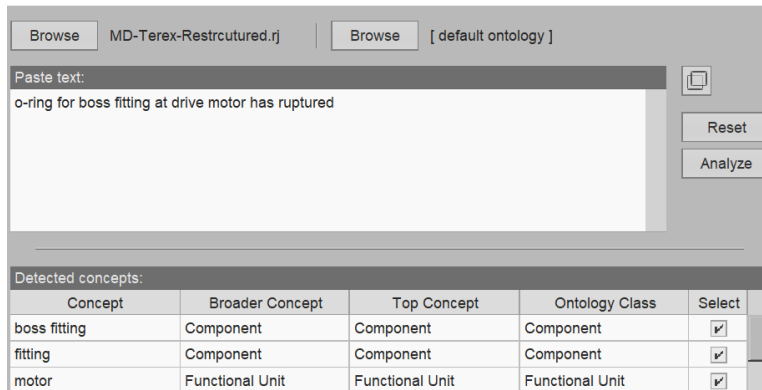


Figure 11. Partial View of Java Tool Showing the Semantic Relationship in Thesaurus.

Mapping properties. The definition of Mapping Properties is similar to the semantic relationship element. We use mapping to express the semantic relationships between concepts in a different concept scheme. Such as for broader and narrower concepts, the SKOS mapping properties are skos:broad match and skos:narrowMatch. To express the inherent meaning for other semantic relationships, these are the following mapping properties: skos:closeMatch, skos:exactMatch, and skos:relatedMatch.

Concept collection. Concepts could have the same label when a group of concepts shares something in common. Such as, in our maintenance, thesaurus, air gun, pump, ECM unit have a common label or top concept **Functional Unit**.

SKOS Tool Functions

The INFONEER SKOS Tool consists of different gadgets such as Thesaurus Manager, Term Selector, Entity Extractor, Concept Model Builder, Concept Model Manager, and Capability Scorer, which can be seen in Figure 12. Figure 12 is a general representation of what the INFONEER SKOS tool looks like. A brief description of the Thesaurus Manager, Term Selector, and Entity Extractor has been provided, as these three tabs have been used mostly during the thesaurus development process.

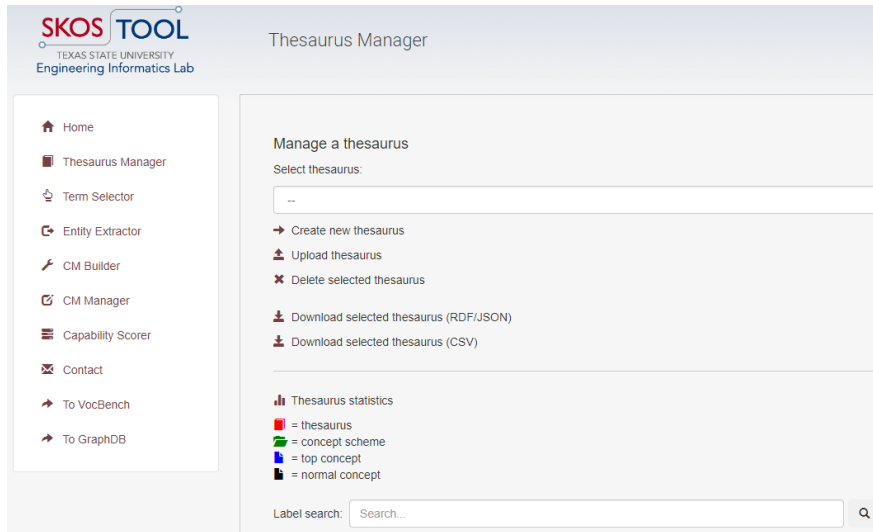


Figure 12. The Opening View of the INFONEER SKOS Tool.

By developing a taxonomy of concepts, adding the appropriate preferred and alternative labels, defining each concept's natural language description, and connecting them to one another, we have built our thesaurus from scratch using the **Thesaurus Manager** tab. We have extended the thesaurus over time by adding more relevant concepts and vocabulary. However, the tab **Term Selector** and **Entity Extractor** were vastly useful when we needed to deal with a larger dataset and wanted to avoid any repetitive words in the thesaurus. In the **Term Selector** tab, we simply inserted raw text and added a new concept under a parent concept. The input text can be inserted directly; the raw data CSV file can be uploaded or grabbed from a given URL. We have inserted some lines of our maintenance raw text in the text box in Figure 13. The terms which are not highlighted mean they are yet not added to the thesaurus under any concept; green highlighted words are the existing ones in the thesaurus, and red highlighted means they are listed as an alternate label of a preferred label.

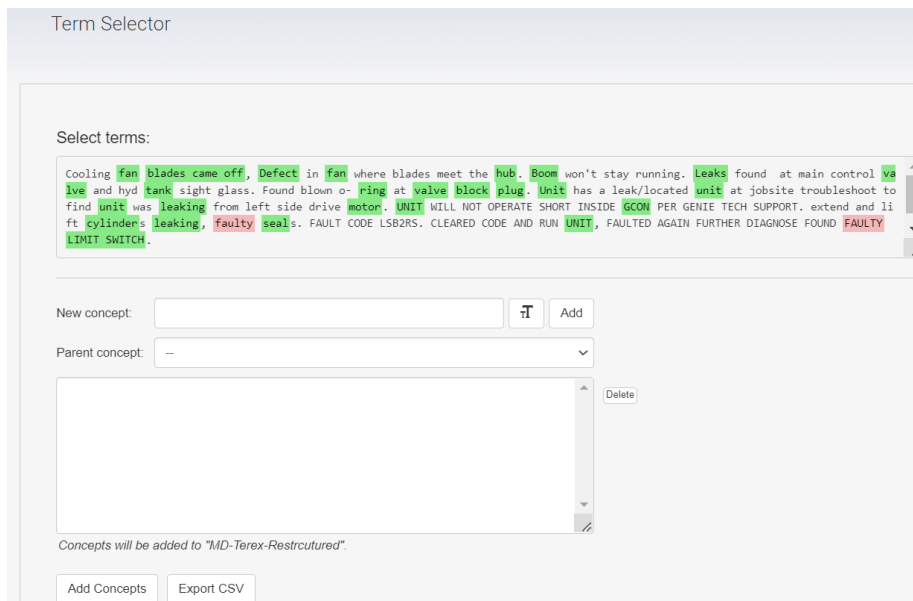


Figure 13. Term Selector View with Example Raw Text in SKOS Tool.

Entity extractor, on the other hand, is being used for tokenizing a text or document. Similar to the term selector, it highlights the words and shows how many times a word has been repeated in the inserted text laid out in Figure 14. The unstructured text had been vectorized as a result, and the concept vector that results had been downloaded as a CSV file. Advanced text analytics procedures like document categorization and clustering can be performed using each document's concept vector. After we have finished building the thesaurus, the output in rdf/json format has been exported to use in the next.

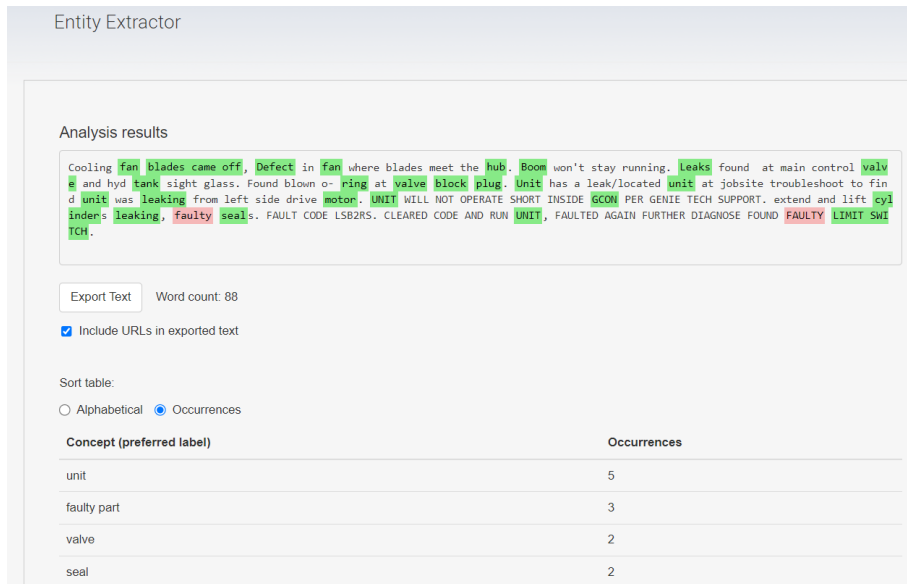


Figure 14. Entity Extractor View with Example Raw Text in SKOS Tool.

Nestor Tool Experimentation

SKOS is both a tool and platform which can be used for data classification under concepts and where the classified data results in an ultimate controlled maintenance vocabulary or thesaurus. As previously mentioned, we moved forward with a hybrid approach to extend the base thesaurus model, meaning we utilized NLP on top of the SKOS tool for a 2nd level classification. Nestor is a free NLP-based TLP toolkit that helps domain experts annotate their Maintenance Work Order (MWO) data through a process called **tagging** (Sexton & Brundage, 2019). Nestor's goal is to assist analysts in making their unstructured, frequently technical, jargon-filled, misspelled, and abbreviated natural language data computable to enhance analysis (Nestor, 2020/2021). Let us take a quick look at the tool in Figure 15 before going into more detail.

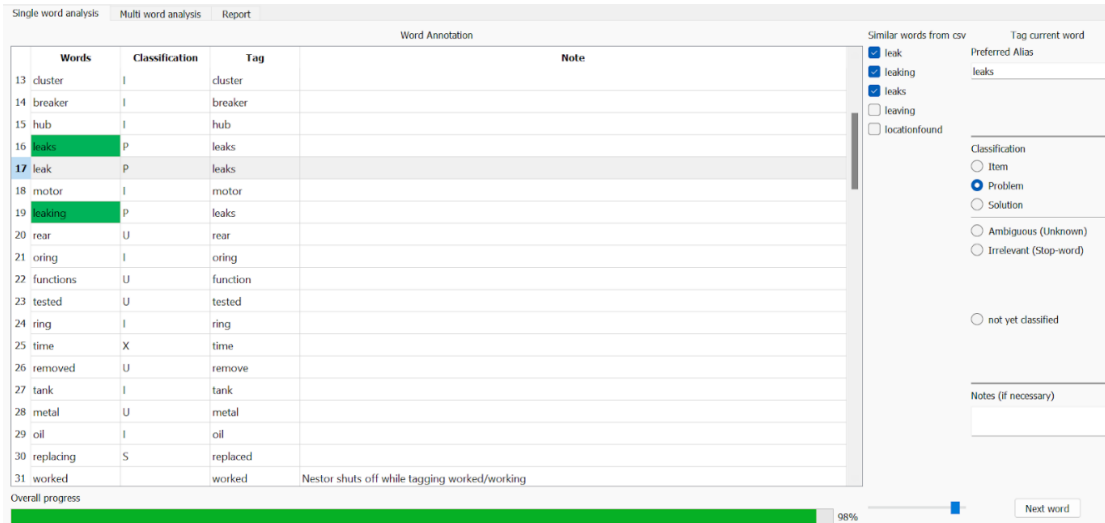


Figure 15. Partial View of Nestor Tool used for Data Tagging.

The raw CSV file can be loaded at first, and the tool automatically identifies words that might be necessary for the classification. The tool can perform both single and multi-word analyses. In the single-word analysis, the tool is internally built with three primaries (**Item, Problem, and Solution**) and two auxiliary (**Ambiguous and Irrelevant**) classifications. However, in the multi-word analysis, instead of having an item and problem, the tool has **Problem Item and Solution Item**. The other classifications are as same as single-word analysis. An example of these classifications is given below in Table 4. Once the words had been determined, we went through each word and manually tagged them under these classifications. Each time we loaded 20 datasets to have a quicker tagging of the words. One of the drawbacks of this tool is that it almost identifies every word and so, it was very time consuming to only load the data first let alone the tagging process. Furthermore, this tool has an option of increasing the sensitivity and so, it would find all the similar word and those can be tagged together. Such as: **leaks, leak, leaking** all are classified as problem but if the sensitivity is increased, the tool would find words like **leaving, locationfound** and those relevant rows

will turn green so that the user understands that those cells have been tagged. The software application will automatically annotate the dataset after the user has finished, and it will then give a CSV file with annotations that may be used as an input for the SKOS thesaurus. Besides, this classification result can be used for failure prediction to reduce breakdowns. The input is for the 2nd level tagging, and by uploading the CSV file under the **entity extractor**, we identified the items, problems, and problem items that we might have missed during the tokenization process.

Table 4. Test Maintenance Raw Text Input and Subsequent Outputs Identified by Nestor

Raw Text	Item (s)	Problem (s)	Solution (s)	Problem Item(s)	Solution Item(s)	Ambiguous	Irrelevant
HYDRAULIC GENERATOR IS LEAKING OUT SHAFT SEAL. SHAFT SEAL FAILURE. REQUESTED SERVICE. P/N 89065GT. REPLACED SEAL	generator , seal, shaft	leaking, failure	replaced	generator leaking, seal failure	replaced seal	P/N 89065GT	requested , service, requested service

Task 2: MWOO Ontology

From the name, it could seem confusing, but we just simply named our owl ontology framework as Maintenance Work Order OWL Ontology (MWOO). Our Ontology consists of Individuals, Properties, and Classes, which roughly correspond to Portege Instances, Slots, and Classes. A domain is described in terms of an OWL ontology, which may contain annotations of classes and individuals as well as detailed descriptions of the properties of those objects. Such as maintenance technician is an object under the agent, whereas the semi-formal Natural Language Definition of the

agent is "A person who is bearer of a Maintenance Technician Role." Before going into the more technical section, let us discuss how an OWL ontology can be developed using Protégé and then we will discuss the components of OWL with respect to Protégé.

Building OWL Ontology

At first, we created a new ontology in Protégé and replaced the default URI with <http://infoneer.txstate.edu/ontology/MWOO>. Under the entity tab, we have added the classes and subclasses of classes in a way that the thesaurus could be mapped with the ontology. For better clarification, Figure 16 is provided for easy interpretation of the mapping from SKOS to Owl. Remember, we want to create a semantic relationship among the concepts of the thesaurus, and so we have created our class hierarchy or taxonomy complementary to the thesaurus.

MD-Terex-Restructured (7)	Mapping to MWOO Ontology class
Artifact (3)	
Asset (18)	Machine
Component (115)	Component
Functional Unit (29)	Functional Unit
Condition (2)	
Conforming Condition (2)	
Conforming Functional Condition (6)	
Conforming Physical Condition (2)	
Nonconforming Condition (2)	
Nonconforming Functional Condition (6)	
Artifact with Incomplete Function (1)	Artifact with Degraded Function
Artifact with Missing Function (9)	Defunct Artifact
Artifact with Undesirable Behaviour (5)	Artifact with Undesirable Behavior
Incomplete Functioning (8)	Degraded Functioning Process
Missing Function (34)	
Undesirable Behavior (31)	Undesirable Behavior
Nonconforming Physical Condition (4)	
Event (1)	
Failure Event (5)	Failure Event
Function (1)	Artifact Function
Maintenance Treatment (8)	
Material Substance (5)	Portion of Material
Property (3)	Quality

Figure 16. SKOS Thesaurus Concept Mapping in Protégé Owl Ontology.

Figure 17 resembles how the owl ontology looks like as well as the classes and subclasses of our maintenance ontology. We named our maintenance ontology

Maintenance Work Order Ontology (MWOO), and we are going to use this name for further referring to the ontology. The 1st class, **OWL Thing**, is a built-in class, and all our newly added classes are subsets of this **OWL Thing**. Two types of classes that could be added are **Subclasses** and **Sibling Classes**. Sibling classes are the individuals which are vertically on the same line in our ontology and are subclasses of the **OWL Thing** as well as the top classes. In this stage, providing an example would be a better approach to having a coherent idea of the individual classes. Classes like Agent, Artifact, Attribute, and Defect are on the same vertical dotted line, and so they are the subclasses of **OWL Thing**. However, these classes may or may not have subclasses under them. Such as, Artifact has three subclasses: Component, Functional Unit, and Machine, and they are sibling classes to each other.

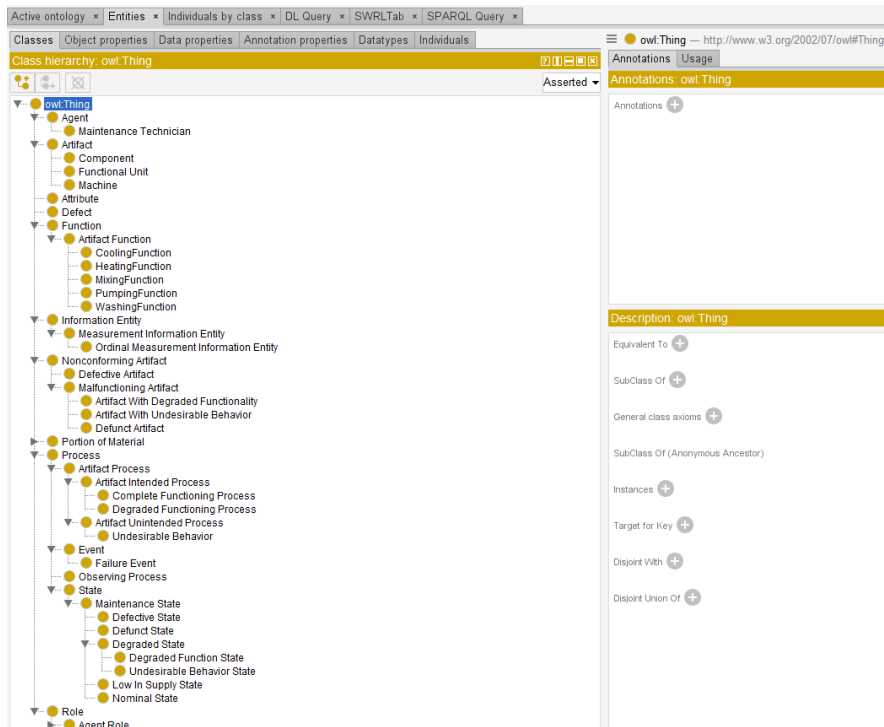


Figure 17. Protégé MWOO Ontology Class View.

Annotations of each class and subclass might have been provided at the right-side box depending on the need, and a general description of the individuals has been provided under the description. We have used the **Disjoint With** for some of the subclasses to ensure that the instances of that individual class only belong to that class. Such as, the cooling function disjoints with the heating, mixing, pumping, and washing function. It will not make sense if one instance is a subclass of multiple groups. Since OWL Classes tend to overlap, we used the disjoint class function when it was absolutely necessary. Now comes the most crucial part of making the Ontology, which is adding the relationship or predicate, which will add the semantic meaning on top of the SKOS thesaurus. Object properties and Datatype properties are the two basic categories of properties. Starting with the **Object Properties** tab presented in Figure 18, we have added different properties and sub properties that have been later used to create the triples relationship while building the knowledge graph. Again, we have provided annotations, including examples, notes, and elucidation for most of the object properties. Domain and range under the description box explain the appropriate triple relation between the subject and object. Each object property may have a corresponding inverse property. Such as a component is a **part of** a machine. This could also be written as the machine **has part** component. As a result, **has part** and **part of** share an inverse relationship.

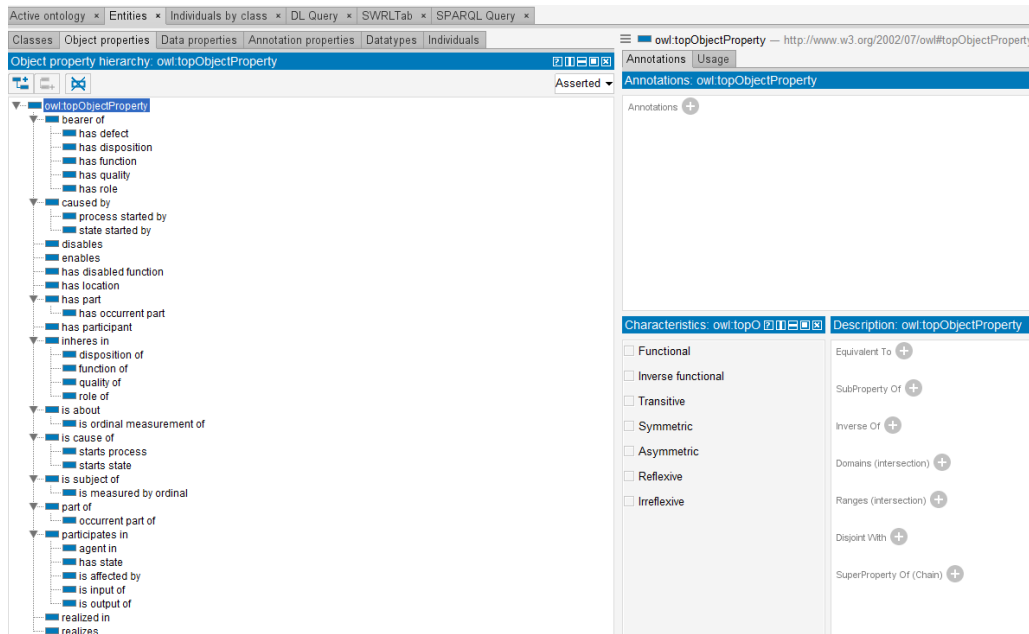


Figure 18. Protégé MWOO Properties View.

Individuals. Individuals are the objects under the classes. Instances are another name for individuals. Individuals can be regarded as **instances of classes** or the subclass of class. In OWL, it must be explicitly specified whether two individuals are the same as one or different from one another; otherwise, they may be defined as the same, or the opposite may be true. Each object has multiple instances coming from the Test Maintenance Company workorders. Let us consider an example below in Figure 19. Functional Unit is a sub-class of the class Artifact. A portion of examples for the instances of **Artifact** class which could be seen in the below figure, are ‘control box’, ‘drive motor’, ‘Drive Selector’, ‘elect motor’ based on Test Maintenance data set. One might be confused as to why some of the instances are repeating themselves. The answer is the repeating instances are coming from different workorders. The two instances of the control box are from workorder 18 and 136 in the Test Maintenance data. Those instances are uniquely identified by distinct URIs even though they have similar labels.

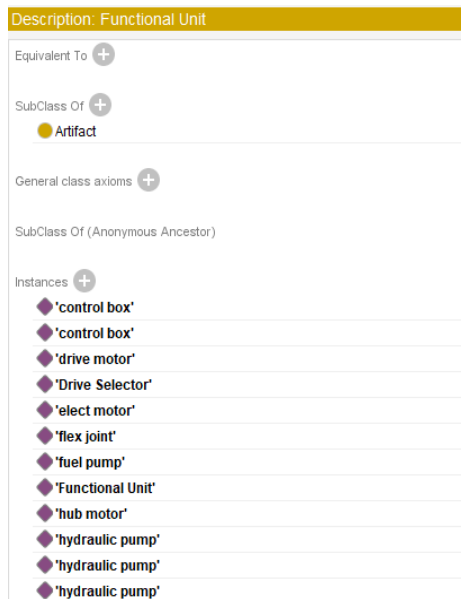


Figure 19. Example of Artifact Instances in the MWO.

Properties. Properties link two individuals together, or if we want to compare it with the RDF triples structure, properties are the predicates. There are two types of properties in OWL: 1) object property and 2) data property. Object properties are the ones that accept only instances at their range, while data properties accept literals (such as strings, numbers, and date-time values) at their range. For example, the property 'has part' is an example of object property because it links the instances of class Artifacts. In Protégé, **Object Properties** is a different tab, and by clicking on each, the annotation characteristics and description can be found. Figure 20 shows most of the object properties we have used to link the individuals and create our triple structure. Remember the example of the control box under **Individuals**, and if we click on the control box of workorder 18, it shows the property relationship '**has part**' and '**has state**'. If we want to explain the properties in words, it would be: control box has a faulty part which is the board and control box is not operating which is expressed by the has state property. Annotation is being provided for '**has state**' so that the meaning of the relationship is

clearly understood before making the triples. Besides, domains and ranges are given to clarify between which instances the **has state** relationship is feasible.

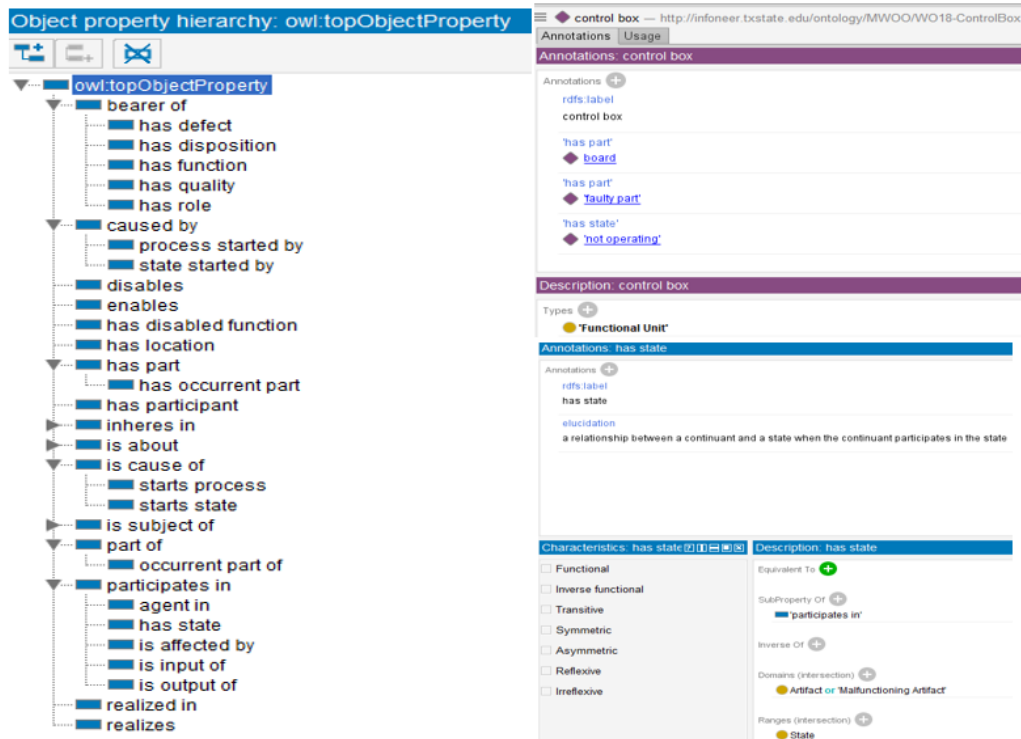


Figure 20. Control Box Property Relations in the MWOO Ontology.

Classes. OWL classes are read as sets with individuals within them. The concepts of thesaurus are mapped under the classes of OWL ontology. Due to that, the word concept is sometimes used in place of class. Classes are a concrete representation of concepts. Formal descriptions that explicitly define the conditions for class membership are used to characterize them. For example, in Figure 21 the class 'Portion of Material' in our ontology would contain all the individuals that are the portion of material in our domain of interest. This 'Portion of Material' has its own fourteen instances, such as hydraulic fluid, air, fluid etc., and two narrower subclasses Oil and Water.

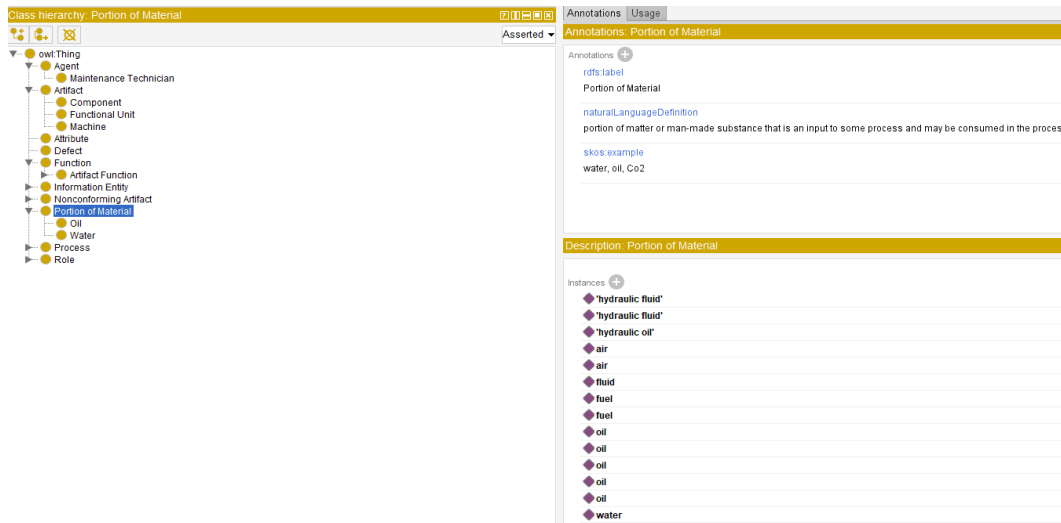


Figure 21. Portion of Material Class Annotation and Description in MWOO.

OWL Ontology Definitions

Before visualizing the OWL ontology graphs, we need to know some of the definitions which we have used more frequently to elucidate some of the examples in the OWL Ontology Visualization coming up soon in the next passage. These definitions are called **Natural Language Definitions**, and human generated. These definitions make ontology easier to understand. In Table 5, the most used **Class/Subclass** definitions are provided, and in Table 6, the frequently used data properties have been defined.

Table 5. Natural Language Definition of MWOO Ontology Classes

Class/Subclass Name	Natural Language Definition
Artifact	object designed by some person or organization to realize a certain function
Component	A part or subassembly, that is intended to become part of a higher-level functional unit, or assembly, or machine, or the final product
Functional Unit	An Artifact that has one or more specific functions and is composed of multiple components and is intended to become part of a machine or equipment.
Machine	man-made artifact containing a set of physically connected components that work together as a unit to realize some intended function.
Defect	An attribute, characteristics, or feature inhered in some Artifact that does not conform to the Design Specifications of the Artifact.

Artifact Function	A Function that inheres in an Artifact and is the primary reason for the existence of the artifact.
Nonconforming Artifact	An Artifact that participates in Defective State or Degraded Function State or Defunct State.
Defective Artifact	An Artifact that is bearer of one or more defects.
Malfunctioning Artifact	An artifact that has a missing function or a function that is partially realized or demonstrates some undesirable behavior
Artifact with Undesirable Behavior	An artifact that realizes some unintended and undesirable dispositions.
Portion of Material	portion of matter or man-made substance that is an input to some process and may be consumed in the process.
Process	p is a process =Def p is an occurrent that has some temporal proper part and for some time t, p has some material entity as participant
Degraded Functioning Process	An Artifact Functioning Process that is the partial realization of an intended function of some artifact.
Undesirable Behavior	An Artifact Unintended Process that causes some undesirable consequences.
Event	An Event that initiates a Defective State or Defunct State or Degraded State
State	A Process in which one or more independent continuants endure in an unchanging condition.
Defective State	A State that holds during a Temporal Interval when an Artifact is bearer of one or more Defects.
Defunct State	A State that holds during a Temporal Interval when an Artifact no longer maintains its designed set of Functions (or at least no longer maintains its primary functions).
Degraded State	A State of an artifact when the artifact function is realized at a degraded level of performance or when some undesirable disposition of the artifact is realized.

Table 6. Natural Language Definition of MWOO Object Properties

Property Name	Natural Language Definition
Bearer of	b bearer of c =Def c inheres in b
Caused by	this relationship is the general form of causal relationship when an entity (continuant or occurrent) is caused by/a result of/a consequence of another entity (continuant or occurrent).
Disables	x disables y when x is a state and y is a function of an artifact z and the function cannot be realized if artifact z is in state x.
Enables	x enables y when x is a state and y is a function of an artifact z and the function can be realized if artifact z is in state x.

Has disabled function	a relationship between an artifact and one of its designed functions when the function is disabled due to the state the artifact has.
Has location	this relationship is between a process or object and the location of the process or object. The location can be absolute or relative (relative to another object).
Has part	a relation that holds between a whole and its part
Is cause of	this relationship is the general form of causal relationship when an entity (continuant or occurrent) is a cause of another entity (continuant or occurrent).
Part of	a relation between a part and a whole
Participates in	a relation between a continuant and a process, in which the continuant is somehow involved in the process
Has state	a relationship between a continuant and a state when the continuant participates in the state
Has ambient condition	a relationship between an entity and the environmental conditions in which the entity exists.

Owl Ontology Visuals

We have built the entire MWOO ontology depending on the thesaurus. In this stage, we understand the very basic that ontology gives meaning to the thesaurus by enabling relationships among instances. Now we will share a few examples of the relationship among those instances, and the way ontology puts a semantic layer on top of the thesaurus, which results in a network of relationships. We will start with simpler examples followed by a bit more detailed ones to allow for better understanding. Figure 22 represents three simpler examples of our MWOO ontology. The yellow color is for the concept classes, and the purple is for the instances under those classes. The upper left side of the figure shows that the class participates in undesirable behavior. **Unit** and **Pump** are the instances of class **Machine** that participates in **Hydraulic Leak** and **Erratic Action**, which are instances of class **Undesirable Behavior**. In our thesaurus, we only had these classes and the instances listed, whereas ontology uses properties like **participates in** to build the semantic relationships between instances. The right side of the figure shows an example raw text: **Basket Not leveling**. It means the level function of

basket has been disabled. It also infers two other relationships: that the basket has defunct state, meaning basket is no longer maintaining its primary function, and the disabled function leveling is causing the basket to be at defunct state.

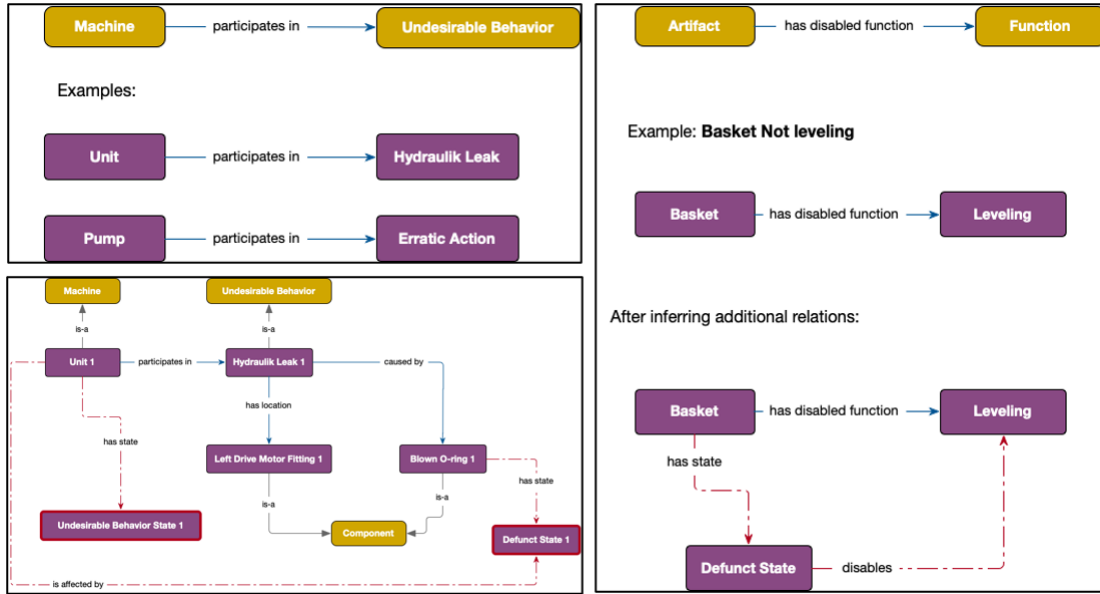


Figure 22. Simple Examples of Ontological Relations in MWO.

The lower left side of Figure 22 explains that **Unit 1**, **Hydraulic Leak 1**, **Left Drive Motor Fitting 1**, and **Blown O-ring 1** are instances of **Machine**, **Undesirable Behavior**, and **Component**, respectively. Unit 1 participates in hydraulic leak, which is caused by a blown O-ring. As a result, the unit has an undesirable behavior state, and blown O-ring has a defunct state. Besides, the undesirable behavior leak has a location which is the drive motor fitting. Nevertheless, it is very difficult and time consuming to infer so many relational meanings only from the raw text and conduct the correct root cause analysis. Now, if we look at Table 6 again, an interesting finding can be seen, and that is the class **State** is a type of **Process** by definition. But why is that? Figure 23 will add some clarity to the State-Process definition and answer the question asked. The top section of the diagram states that the failure process is a Change and is initiated by an

External Event. When a process fails, it will result in a **Failure Event** and will make the **Artifact** go into a **State**. Simply put, a failure event and state are both processes, the process initiates a failure event, and the state is created.

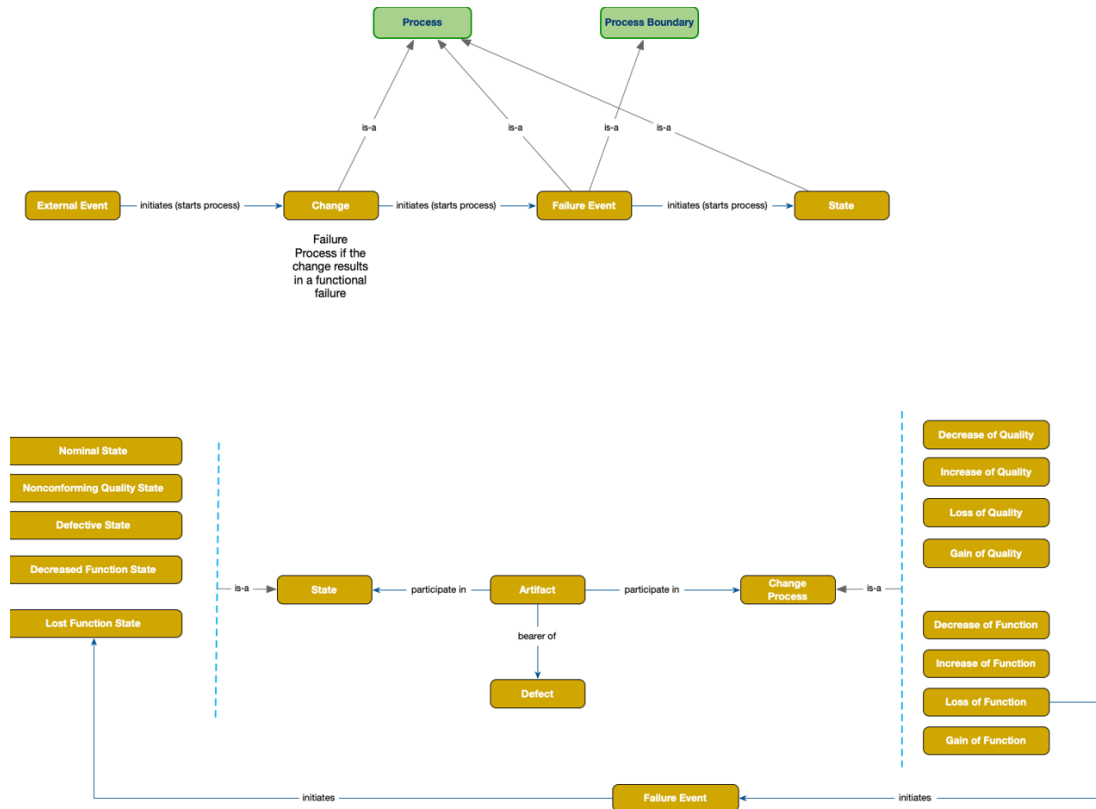


Figure 23. Class State vs Class Process in MWO.

State and process are comprised of other sub-states and sub-processes. The bottom section refers that **Artifact** is a bearer of **Defect** and could participate in several processes. These processes start the failure events, which in turn result in different states. Hence, we can conclude that the Artifact participates in several state, including **Nominal State**, **Defective State**, **Lost Function State** and so on. Table 7 will be helpful in understanding the basic difference between state and process in our maintenance ontology. Two examples from the Test Maintenance have been typed in to reveal the distinction. In the first example, unit has a state which is **not starting**. A defective artifact

initiates the state; there is no change process to trigger the initiation of state. However, in the 2nd example, **leaking** is a change in process, and the hydraulic pump participates in leaking and has obtained the state **not starting**. That leaking is caused by a broken seal which is a defective artifact.

Table 7. Examples of State and Process in MWOO Ontology

Num	Problem Statements	Artifact	State	Process	Defective Artifact
1.	Unit not starting. Found dead battery.	unit, battery	not starting		dead battery
2.	Hydraulic pump is leaking, not starting. Found broken seal.	hydraulic pump, seal	not starting	leaking	broken seal

Our last example in Figure 24 is just a bit complex one with more semantic relationships among the individuals, and we will outline each relationship. In fact, this example works as our guideline for creating triples relationship between two instances in other examples. The machine is a subclass of **Artifact**; the functional unit and component are part of machines. So, we have used **part** data properties among them to express the internal relation. Five relation linkages evolve around **Artifact**, and we have bulleted them below to be coherent with the content of the figure.

- Artifact is a bearer of **Defect**
- Artifact participates in **Process**
- Artifact is located in an **Operational Area** that has an **Ambient Condition**
- Artifact has **Function** which could be a disabled function
- Artifact has **State** which is both caused by the **Process or Defect** and other **Artifact or Defective Artifact**. The **Defect** and **Defective Artifact** have a location and lastly, the **State** either enables or disables the **Function**.

The upper mentioned classes and object relationships have been mostly used to construct the knowledge graphs for our Test Maintenance Dataset. As a result, it was essential to picture the relationships beforehand we moved forward to our Task 3. The outcomes of these ontological relationships grow into a vast number of graphs, namely Knowledge Graph, which we will elaborate in the next paragraph.

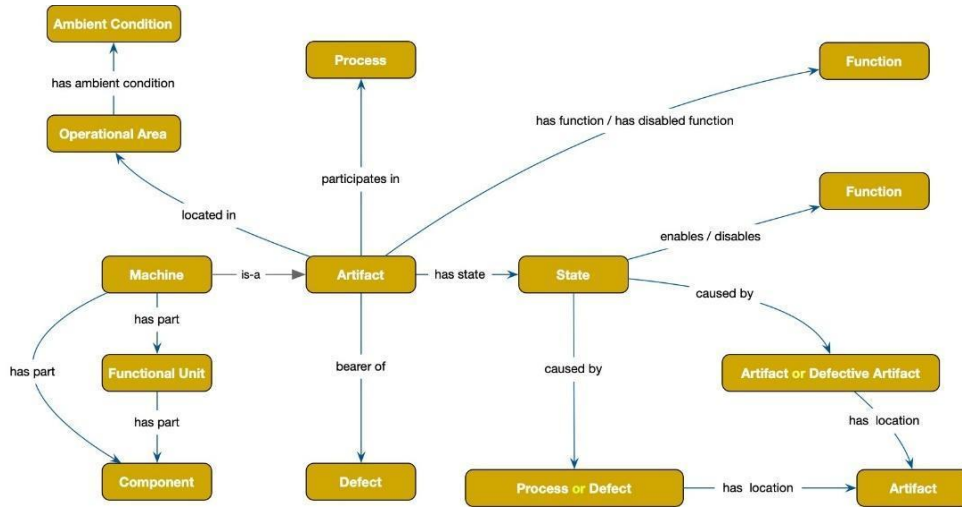


Figure 24. Complex Example of Ontological Relations in MWOO Ontology.

Task 3: Knowledge Graph Generation

We have already manifested how the ontology looks with the semantic relationships among the instances of the Test Maintenance Company raw texts. We have generated the relational figures using Microsoft Office tools while describing the ontology in Task 2. Now we will introduce the tool which we have used for the actual visualization of the ontology. Our developed ontology is a storehouse of all the mapped concepts from the SKOS thesaurus and object properties. We displayed a glimpse of our Maintenance Annotation Tool in Figure 11 while discussing the semantic relationship element of the SKOS tool. This java based tool is called **Maintenance Workorder Annotation Tool (MWOAT)**. This java based tool uses the SKOS thesaurus and OWL

ontology as the input to generate the knowledge graphs. Figure 25 is our annotation tool view, and different tab sections can be seen to initiate the graph generation. The left side of the figure is the 1st part of the tool, and the right side is the 2nd page.

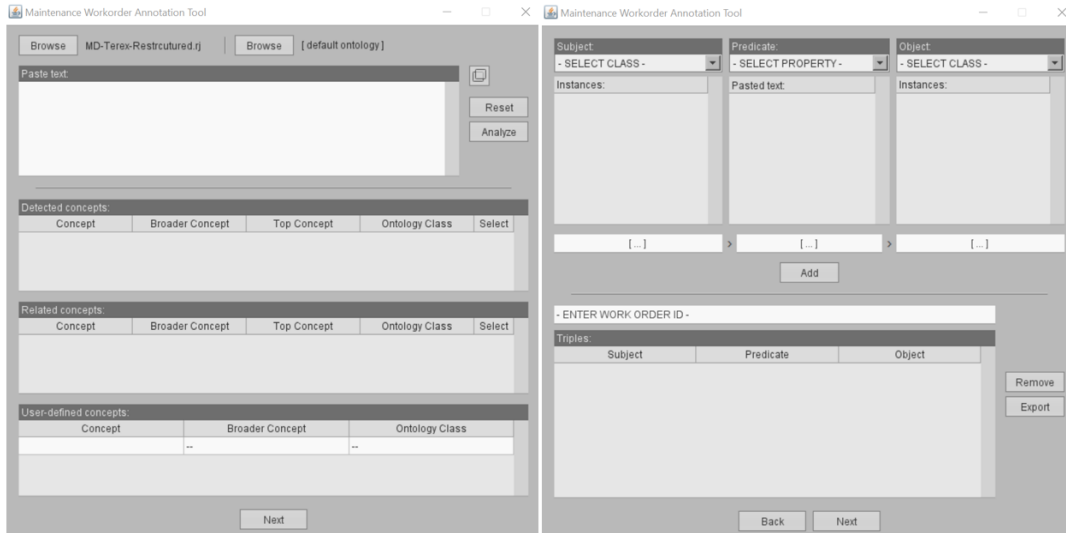


Figure 25. Maintenance Workorder Annotation Tool (MWOAT) Tool Full View.

The ontology we generated as a previous task is integrated into the tool and we load the maintenance thesaurus each time, so if needed we can also analyze other datasets with our developed ontology. In the paste text section, we copy and paste each workorder from the Test Maintenance dataset and analyze it to detect the concept. By detecting, we meant the tool detects the concept words in the work orders along with their broader and top concept that exists in the thesaurus and have been mapped to ontology. The tool also finds the related concepts in the same way, to provide us with extended options to show on the graph. The tool is able to find both single and multi-words from the thesaurus. As a result, the users will decide if they want to keep the single word, multi-word, or both concepts as well as how they want to model it. Such as for Broken O-ring, the tool will detect the O-ring as a **Component**, Broken as **State**, and **Broken O-ring** as **Defective Artifact**. Now one can only select components and state and build the triple relationship

between them, option two is to select **Component** and **Defective Artifact**, and the final option is select all of them to generate as many as relationship possible to convey clear meanings.

Furthermore, we work with large data sets; sometimes, it is possible that we might have missed tagging some individuals. But one of the tool's best features is that we can tag the un-tagged words as concepts under the **Broader Concept** of thesaurus and map them in the **Ontology Class** by using the User-defined concepts for the sake of building the knowledge graph on the next page. The newly added concepts in the tool do not automatically update in thesaurus or ontology, and the user has to go back and put them in both places. However, it makes users' jobs a bit easy as they do not have to update the changes every time they find something new. They can just move forward with generating the knowledge graphs and simply make a list of the untagged concept to tag them altogether later. Once they go through the detected concepts and make essential changes, they would hit **Next** and go to the 2nd page to build the triples relationship among the subject, predicate, and concept.

In the mid column under **Pasted Text**, the work order text will automatically appear from the 1st page as well as the object properties from the Owl ontology. Remember the previously mentioned definition of Owl Properties "Properties link two individuals together, or if we want to compare it with the RDF triples structure, properties are the predicates." The linkable individuals are the subject and object here, and the detected concepts will be clickable in both the 1st column and the 3rd column. The user will connect different subjects and objects using appropriate data properties to create multiple numbers of the RDF triples and add them in the below space of the tool. Once

all the triples are generated, they will put the workorder ID in the designated box and export the output in text form, which is our knowledge graph. Now, if the user wants to visualize the knowledge graph, a web service called RDF Grapher (<https://www.ldf.fi/service/rdf-grapher>) allows the user to paste the text form of the output and generate an image of the graph. The image provides an easy interpretation of the graph, and it serves as a cause diagram by showing the co-relation among individuals.

Example 1

In this section, a few examples from our Test dataset will be examined to see how the tool looks like when it detects the concepts and eventually generates the knowledge graphs. The first example of raw text is “*Equipment will not move. Boom Basket won’t rotate to the left. Needs Rotator.*” We paste the text in the paste box, and when we hit **analyze**, it detects the concepts that are already in the thesaurus. The detected concept consists of an instance's broader and top concept in the thesaurus and the ontology class in OWL. This lets the user see how a concept is mapped and if the detected concept is integral for the knowledge graph generation. As aforementioned, the tool will detect both single and multi phrases, and we can just unselect the recurrent concepts. For this example, the tool did not detect any repeating or related concepts, so we selected all the concepts identified by the tool. The leftmost and middle portion of Figure 26 shows the detected concept, such as Basket is a Component; Equipment is an Asset in the thesaurus, and an instance of the class Machine in the ontology. Not Moving is Nonconforming Condition in the thesaurus and the Defunct State class instance in the ontology. Now, one can question why concept classes of Equipment and Not Moving are not similar like

basket. The easiest answer would be SKOS tool is company centric, and we want them as informative as it works as it can be treated as lightweight ontology. At the same time, using both the SKOS concept scheme and Owl ontology results in developing generic entities that can be applied to a wide range of maintenance applications and extended to meet specific use cases beyond our project. So, we can conclude that Machine and Defunct State are generic concepts in ontology and differ from the thesaurus class. This is the main reason why we map the SKOS concept in OWL ontology.

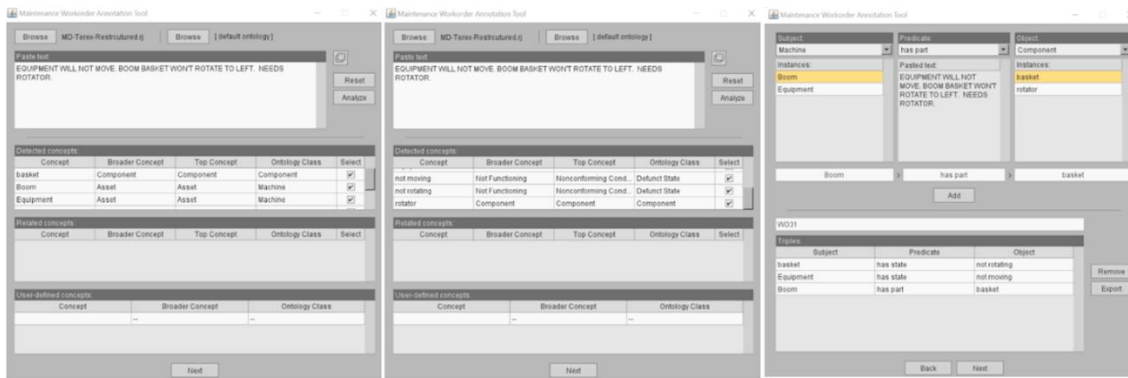


Figure 26. Example 1 Analysis in MWOAT Tool.

Now moving forward with our example, the right side of Figure 26 explains the semantic relationship between the subjects and predicates. We chose from the drop-down list to generate the meaningful semantic triple relationships we wanted to visualize in our knowledge graph and added them one by one. Such as, Not Moving is a State, and so we created equipment **has state** Not Moving. Also, the annotations, if the predicate or the object property is correct or not coming out from the subject or domain, are provided within the Owl Ontology. A domain is a class to which the subject of an RDF statement using a given property belongs, and a range is the class of its object (value). The domain value restricts the class of the subject while formulating a triple structure, and the range value restricts the range of the property value. Afterward, we exported the result as

WO31 and pasted the exported raw text into the RDF graph to see what we generated.

Figure 27 is the knowledge graph visualization for Example 1. Starting from the left, boom is a type of Machine which is labeled as **Boom** in Ontology. Equipment is a type of machine, too, labeled as **Machine** which has state Not Moving. Basket is a component, labeled as **basket** and since basket is not rotating, it results in Not Rotating state. Not moving and not Rotating both go under the sub-class of state, Defunct State. Rotator is a type of Component, which is labeled as **rotator** in Ontology. Lastly, boom, rotator, basket, equipment; all of them are instances in Owl ontology, and so they are all Owl individuals and linked with owl:NamedIndividual.

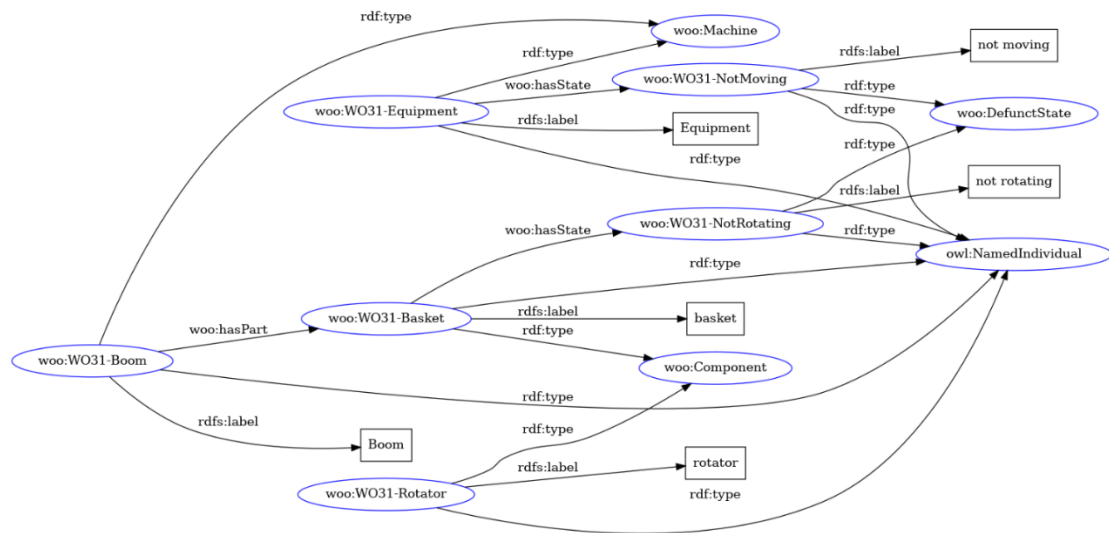


Figure 27. Example 1 Knowledge Graph Visualization in RDF Grapher.

Example 2

The second example shown in Figure 28 is the workorder 27 in our Test Maintenance Dataset, and that is “Unit will not go up at all the way o-ring on manifold leaking Found unit had a hyd leak.” Clearly, the problem statement is full of technical jargon. Our MWOAT tool successfully identified most of the essential concepts, but it identified both the single and multi-phrases. Hence, it is visible under the detected

concepts that it tagged both o-ring and ring. We unselected the ring since both of them were the same. Besides, the tool also found alternate and related concepts, such as the problem statement only had a leaking, which falls under Undesirable Behavior in the thesaurus and is the upper class for leaking oil. Leaking oil is an alternate concept for **leaking**, so the tool listed leaking oil under detected concepts as the tool does not still have any individual box or space for alternate concepts. The oil spill is a subclass of leaking and was entered as related concept of **leaking**, so oil spill is detected as a related concept, but since we do not need it for this workorder graph generation, we simply unchecked it.

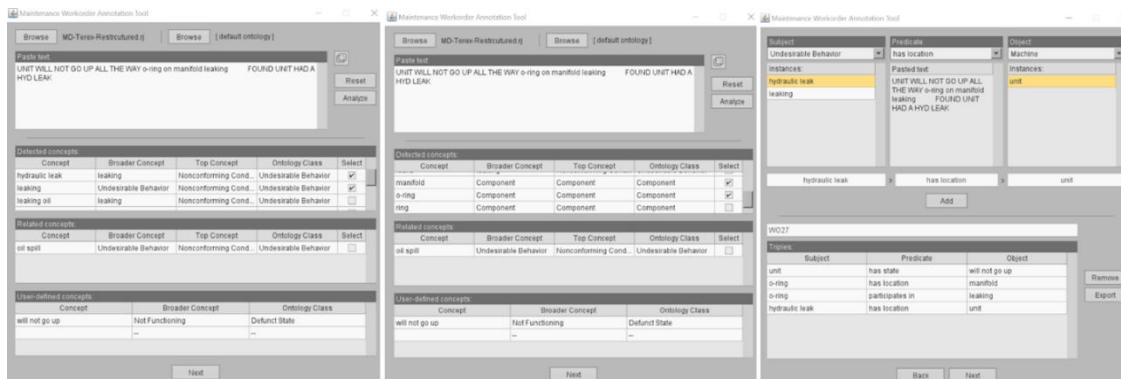


Figure 28. Example 2 Analysis in MWOAT Tool.

The additional concept that we need to emphasize in the 2nd example is the User-defined concept. We did not use **will not go up** under any bucket in the thesaurus, but from the basic understanding, we know that it means similar to not moving or not functioning. As a result, we used **will not go up** for the sake of this workorder knowledge graph generation. It will not be added to the thesaurus or ontology, and the user has to add it manually. Another alternate solution to similar situations where the concepts are untagged is to try finding similar meaning concepts. For instance, we could have simply used **will not lift** instead of **will not go up** and the tool would have identified it, and we

did not have to use **User-defined concepts** then. However, we just stayed with the alternate one, kept the raw text as it was, and added a user-defined concept. The relationship that we defined between the subject and predicate, is similar to the way we did for Example 1. Recall from Chapter 2 that the subject and object change depending on the inner object property relation between them. In the first triple, the Unit is the subject that has a defunct state, whereas, in the last structure, unit is the location of the hydraulic leak and became an object. The knowledge graph explanation of Example 2 is also homogeneous to Example 1 knowledge graph.

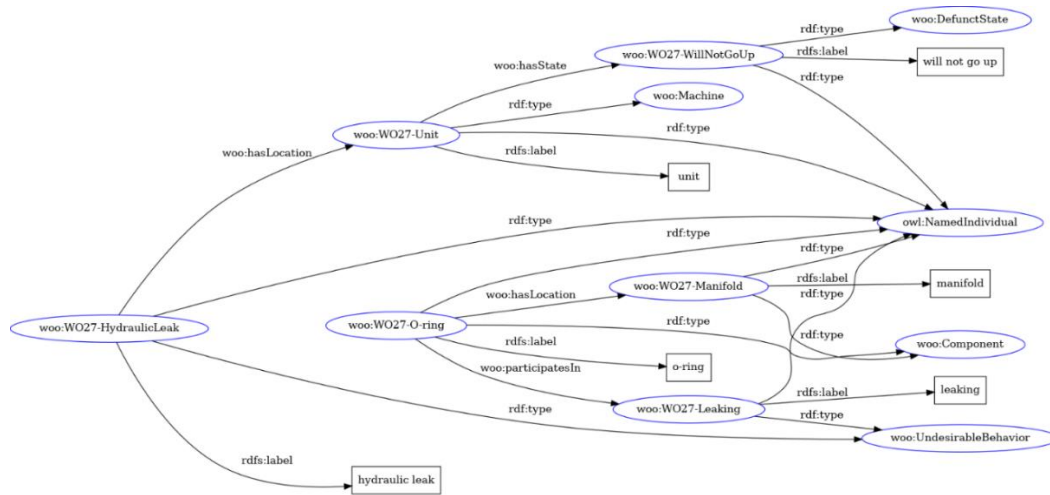


Figure 29. Example 2 Knowledge Graph Visualization in RDF Grapher.

The hydraulic leak is labeled as **hydraulic leak** and located in the Unit. The rdf label for the unit in Owl ontology is **Unit**, which is a type of machine and has the user-defined concept or defunct state. The leaking is undesirable behavior, and the leaking O-ring has a location on the manifold. A quick note to mention here is that **woo** is the prefix name for our maintenance ontology and expands to an IRI (Internationalized Resource Identifier). For our case, it is woo: <http://infoneer.txstate.edu/ontology/MWOO/>, where

woo is the name, and the later portion refers to our ontology IRI.

Example 3

This will be our last example explanation, so let us look at a bigger, complex example and divide it into smaller parts for easy understanding. We will analyze workorder 10 in Figure 30, which states “*front diff is making oil, front diff blew out snap ring internally, removed one side and found diff to be no good snap ring blew apart causing damage to case.*” After pasting the raw text, we realized that a number of concepts were untagged in the thesaurus, but few of them have closer meaning synonyms already present in the thesaurus. As a result, we changed some of the wordings in the text paste box, such as: making →leaking, blew out →blew apart, to be no good →damaged. We also added the user-defined concepts: diffuser, snap ring, and case to be visible in the knowledge graph. The tool identified the ring under the detected concept, but we specifically wanted to define the snap ring for a better root cause analysis of the maintenance problem. We kept the ring just to show it on the graph, as it does not impact the result. The oil spill was also detected like the previous example problem, and we have unchecked it.

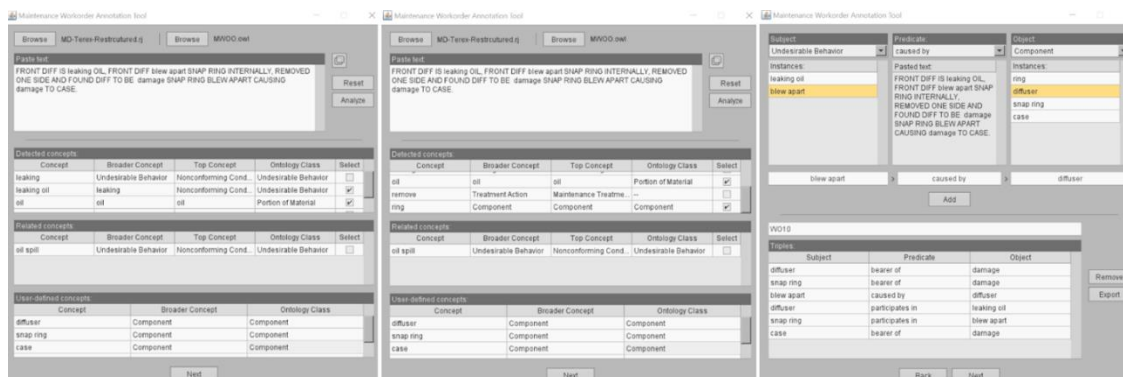


Figure 30. Example 2 Analysis in MWOAT Tool.

The last page of the tool shows the triple relationships generated. The diffuser

participates in leaking oil, and the snap ring participates in blew apart. The statement also states that snap ring internally blew apart is due to damaged diffuser and resulted in a damaged snap ring also. And finally, the case becomes damaged due to the blew apart snap ring. Figure 31 looks complex with all the interconnected links, but it is the exact reflection of the six triple relationships we just created.

Figure 31

Example 3 Knowledge Graph Visualization in RDF Grapher

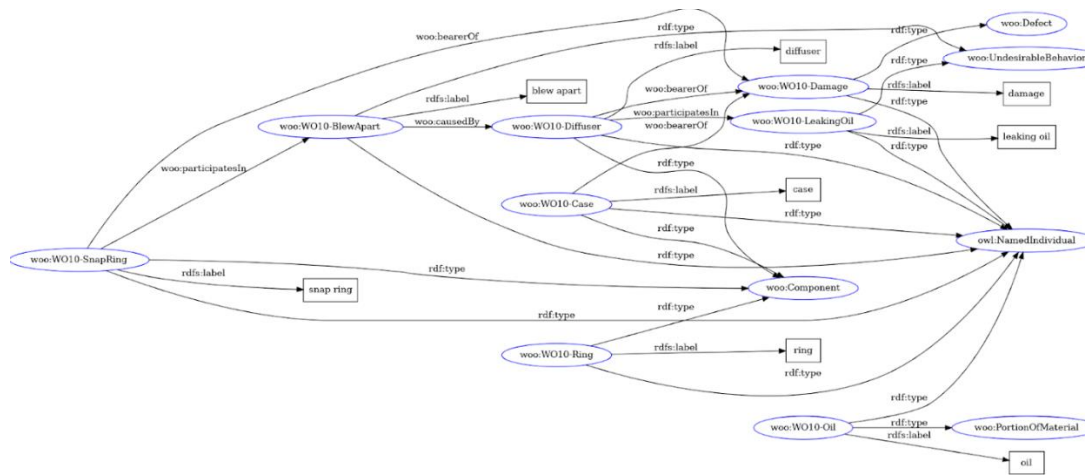


Figure 31. Example 3 Knowledge Graph Visualization in RDF Grapher.

Starting with the component Snap Ring, which participates in Blew Apart and bearer of Damage. Blew Apart is a defect caused by Diffuser; Diffuser participates in Leaking Oil and Damaged as well. The case is also a bearer of damage, and lastly, the ring does not have any relationship with other individuals as we have used the snap ring instead. The knowledge graph can be treated as a powerful tool because it identifies the root cause of a maintenance problem from simple to complex raw text. In addition, it is very difficult to have a concise idea of the failure only from the raw text as they are full of errors and technical vocabulary.

Task 4: Validation

This is the last section of this chapter. In this section, we will illustrate the SPARQL query and how we have utilized it for our fourth task fulfillment. We will also show how the **Reasoner** plug-in on the Protégé tool automatically allows adding instances under different classes. Previously, a brief of SPARQL query has been provided in Chapter 2, where we explained that it is a semantic RDF query language to retrieve required information stored in RDF triple format. Queries could be done on raw data; however, in most cases, raw data needs additional processing in order to be used as information. Raw data lacks the built-in capacity for consistency and querying over the raw data needs reconstructing the schema, which ends up resulting in larger duplicate data. As the quantity and complexity of relationships rise, relationship queries in traditional raw text databases will come to a standstill.

In contrast, the RDF knowledge graph allows new relationships over time without endangering current functionality resulting in more useful query results. Of course, raw data is necessary to trace back the data source, but a knowledge graph is a single place where all data and the interlocking relationships behind that data can be found. This enables one to find information faster, uncover hidden insights into text data, and understand how everything is connected on a broader scale. We require the connected, reusable, and flexible data foundation to reflect the complexity of reality in order to address difficult challenges that require the integration of several unstructured raw text data. Multiple interpretations of the same data can be made possible by connected, meaning-rich data, making it easier to find answers to complex queries and more quickly extract insights. These answers will be helpful to our validation purposes, meaning

depending on the SPARQL query results, we can always go back to our ontology and rule out the detected classes as well as the instances under them to verify the true positive mentioned in chapter 1. We will also elaborate on the process with our query examples in this chapter.

We have used java based **Stardog** (<https://www.stardog.com/>) application platform, which supports SPARQL query for querying RDF knowledge graphs. The most cutting-edge graph data virtualization and high-performance graph database are available from Stardog. Stardog uses the RDF triples knowledge graph and OWL ontology as input. After creating RDF triples for each raw text data in the Test Maintenance Dataset, we have created a master file, which is an accumulation of all the two hundred and fifty RDF triples. We have simply loaded our RDF triples master file on Stardog, written it down, and run the queries to find out the expected information we needed. In Addition, ontology IRI is used at the start of the query writing so that the tool can refer to it. The platform also sends error messages to the user if any structural issue is found in the queries. We have used several queries to validate our model, and a number of queries will be elaborately described below in the next paragraph. As soon as we open the Stardog Studio application, a blank workspace with several tabs appears. Such as in Figure 32, we can see the tool's built-in database is detected as **stardog-tutorial-beatles**. Users can load their work specific datasets, write the queries in the workspace console and run them to retrieve the results. Once run, another console appears below the first console, which shows the result.

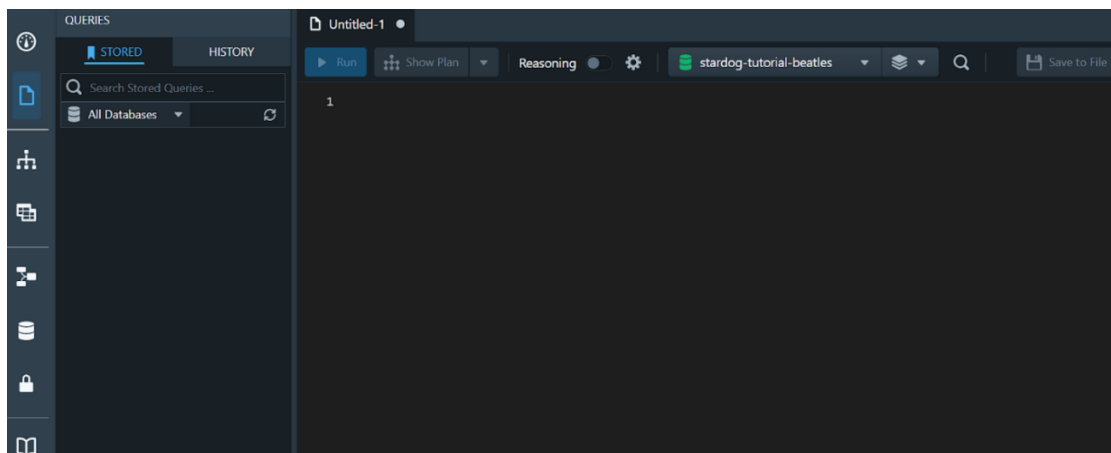


Figure 32. Partial View of Stardog Studio Query Application.

In addition, built-in dataset as well as their tutorials, are available in this application which is specifically useful for the new user to guide them in building queries. Some examples of query operations are SELECT, ASK, DISTINCT, COUNT, LIMIT, ORDER BY, and GROUP BY, and we will break down some of their usage in our maintenance database queries. We will start with the simplest query and move forward with a bit more complex one to provide a better understanding of the query framework. Before jumping into the main query examples, let us get familiarized with the basic query structure which we have used in all our examples. The main query form in SPARQL is a SELECT query which is used to extract results from the knowledge graph triples and can be modified to match the desired pattern. A SELECT query has two main components: a list of selected variables and a WHERE clause for specifying the graph patterns to match. The SELECT structure looks like this,

SELECT <variables>

Where{

<graph-patterns>

} , where the variables are the output, and the graph patterns are descriptions to match

Triple patterns are the fundamental building component for SPARQL searches. A triple pattern is identical to an RDF graph triple, with the exception that any one of the three positions can include a variable after the SELECT command. We search the knowledge graphs for matching triples using triple patterns, and variables function as wildcards that can match any node. For example, the simplest query to retrieve the subject(?S), predicate (?P), and object (?O) from our knowledge graphs are shown in Figure 33. Here, the SELECT query returns the triples from the maintenance knowledge graphs. The output variables ?S ?P, and ?O tell the SELECT query what to return. The output variables belong to the declared type or label between the WHERE clause, and simply put, the WHERE is used to extract only the records that fulfill a specified condition. In this case, we want to retrieve all the triples structure in our knowledge graph, so we write the variables ?S ?P, and ?O, showing we want triple structures where a subject, predicate, and object exist, with any value in any position. This simplest query retrieves 1000 results in 170 milliseconds in the lower console. We can also visualize each of the results in a graphical mode which we will see with our query examples.

```

1 PREFIX woo: <http://infoneer.txstate.edu/ontology/MWOO/>
2 #Return all the triple structure from the matched structure in knowledge graph
3 SELECT ?s ?p ?o
4 WHERE{
5   |   ?s ?p ?o
6 }

```

Run to File
Text
Charts
Visualize
1,000 Results, 170 ms

s	p	o
http://infoneer.txstate.edu/ontology/MWOO/WO31-Basket	rdf:type	http://infoneer.txstate.edu/ontology/MWOO/Component
http://infoneer.txstate.edu/ontology/MWOO/WO31-Rotator	rdf:type	http://infoneer.txstate.edu/ontology/MWOO/Component
http://infoneer.txstate.edu/ontology/MWOO/WO362-Hose	rdf:type	http://infoneer.txstate.edu/ontology/MWOO/Component
http://infoneer.txstate.edu/ontology/MWOO/WO362-LiftCylinder	rdf:type	http://infoneer.txstate.edu/ontology/MWOO/Component
http://infoneer.txstate.edu/ontology/MWOO/WO3-Charger	rdf:type	http://infoneer.txstate.edu/ontology/MWOO/Component
http://infoneer.txstate.edu/ontology/MWOO/WO4-BossFitting	rdf:type	http://infoneer.txstate.edu/ontology/MWOO/Component
http://infoneer.txstate.edu/ontology/MWOO/WO4-O-ring	rdf:type	http://infoneer.txstate.edu/ontology/MWOO/Component
http://infoneer.txstate.edu/ontology/MWOO/WO5-Switch	rdf:type	http://infoneer.txstate.edu/ontology/MWOO/Component

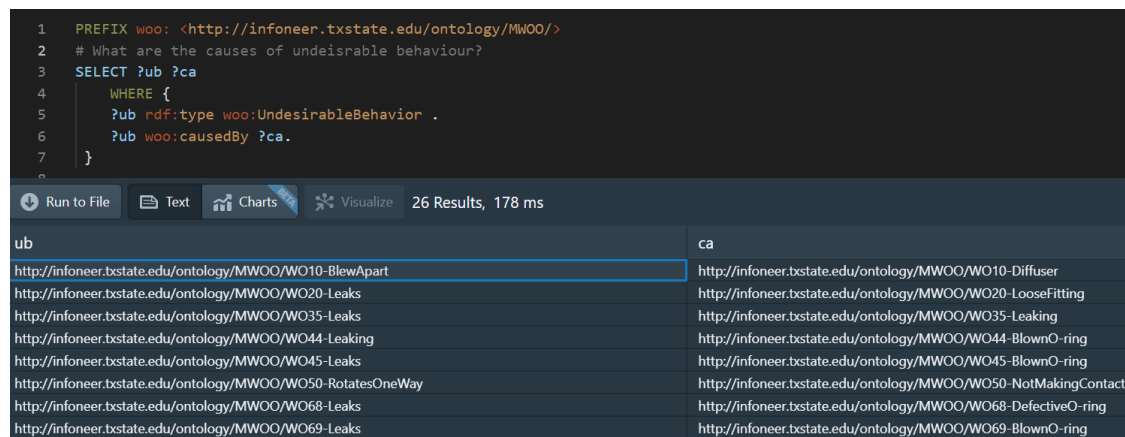
Figure 33. Simplest Query Structure using SELECT Command.

Query Examples

Query Example 1

What are the causes of undesirable behavior?

We want to know what the causes of undesirable behavior from this query are. In other words, how many **caused by** object property relationships we have with the instances under **Undesirable Behavior** class in our knowledge graph. In Figure 34, starting from the 1st line of the query is the declaration of our ontology prefix. Whenever pound (#) sign is put at the beginning of the sentence, it becomes a comment, and we have used it to write down our query in the workspace. We have used **ub** for undesirable behavior and **ca** for caused by as dynamic parameters. A dynamic parameter is a parameter to a SPARQL statement for which the value is not specified when the statement is created. Instead, the statement has a question mark (?) as a placeholder for each dynamic parameter. The rest of the information on the dynamic parameters would be given inside the WHERE clause.



The screenshot shows a SPARQL query editor with a dark theme. The query is as follows:

```
1 PREFIX woo: <http://infooneer.txstate.edu/ontology/MWOO/>
2 # What are the causes of undesirable behaviour?
3 SELECT ?ub ?ca
4 WHERE {
5   ?ub rdf:type woo:UndesirableBehavior .
6   ?ub woo:causedBy ?ca.
7 }
```

Below the query, there is a toolbar with buttons for 'Run to File', 'Text', 'Charts', and 'Visualize'. The status bar indicates '26 Results, 178 ms'. The results are displayed in a table with two columns: 'ub' and 'ca'.

ub	ca
http://infooneer.txstate.edu/ontology/MWOO/WO10-BlewApart	http://infooneer.txstate.edu/ontology/MWOO/WO10-Diffuser
http://infooneer.txstate.edu/ontology/MWOO/WO20-Leaks	http://infooneer.txstate.edu/ontology/MWOO/WO20-LooseFitting
http://infooneer.txstate.edu/ontology/MWOO/WO35-Leaks	http://infooneer.txstate.edu/ontology/MWOO/WO35-Leaking
http://infooneer.txstate.edu/ontology/MWOO/WO44-Leaking	http://infooneer.txstate.edu/ontology/MWOO/WO44-BlownO-ring
http://infooneer.txstate.edu/ontology/MWOO/WO45-Leaks	http://infooneer.txstate.edu/ontology/MWOO/WO45-BlownO-ring
http://infooneer.txstate.edu/ontology/MWOO/WO50-RotatesOneWay	http://infooneer.txstate.edu/ontology/MWOO/WO50-NotMakingContact
http://infooneer.txstate.edu/ontology/MWOO/WO68-Leaks	http://infooneer.txstate.edu/ontology/MWOO/WO68-DefectiveO-ring
http://infooneer.txstate.edu/ontology/MWOO/WO69-Leaks	http://infooneer.txstate.edu/ontology/MWOO/WO69-BlownO-ring

Figure 34. Query Example 1.

Next, we have told the query a place where the necessary information for **ub** and **ca** could be found. In our ontology, we have the **Undesirable Behavior** class, and so, we

have declared the ub as a `rdf:type` of undesirable behavior. The prefix name is **woo:UndesirableBehavior** and one condition is that the exact prefix name has to be used to get the result. We have to use a dot (.) at the end of each line inside WHERE clause; only the last line can skip the dot. We were only interested in how the undesirable behaviors are caused by other class instances and declared the ontology prefix caused by as **woo:causedBy** and closed the bracket. After that, we closed the bracket and ran the query, which resulted in 26 undesirable behaviors caused by instances of classes. Such as in the 1st line of the results reflected that in our Test maintenance workorder 10, the undesirable behavior **Blew Apart** is caused by **Diffuser**. We can also click on each workorder row to see the result visually and validate it. Let us consider validating the 1st result against the maintenance workorder which was, “*front diff is making oil, front diff blew out snap ring internally, removed one side and found diff to be no good snap ring blew apart causing damage to case.*” The resultant graph page in Figure 35 is very informative as it provides the total number of classes and properties expressed via nodes and edges.

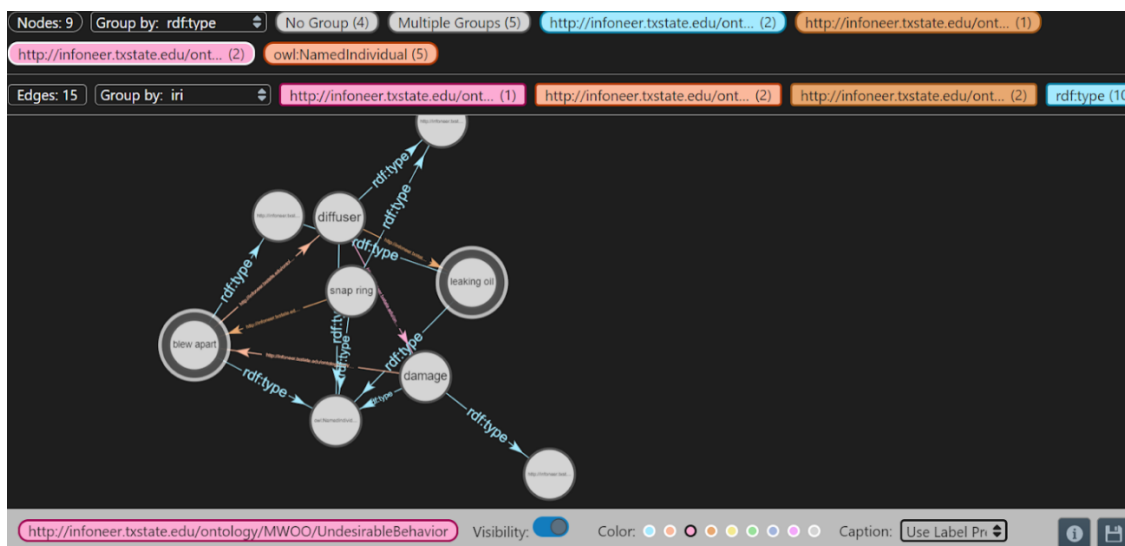


Figure 35. Query Example 1 Result Partial Visual Representation.

For the 1st result workorder, we have a total of 9 nodes and 15 edges. The nodes include the instances of class Component, Undesirable Behavior, and Defect, and these are connected by edges which include the type and the object property. Each of the color-coded tabs, as well as the nodes/edges, could be clicked for the ease of easier root cause of the problem. Such as, we set our cursor on the pink tab in the 2nd row and two nodes ‘blew apart’ and ‘leaking oil’ popped up. So, why our query gave us only one undesirable behavior? Because only blew apart maintains the caused by the relationship with ‘diffuser’ node. On the other hand, diffuser participates in ‘leaking oil’ node and so, the query did not show this triple relation in the result. All of the possible relations in the raw maintenance workorder have been correctly presented in the graph, and after comparing the visual result with the raw text, it is much easier to conclude that the diffuse is the root cause of the failures. This visualization is similar to the RDF Grapher online based tool we have used for our knowledge graph visualization. However, Stardog provides information in much better way and with color-coded nodes/edges. Since the graphs tend to have a lot of semantic relations, analyzing the root cause of failure using Stardog is a much-preferred way.

Another interesting addition in query 1 would be adding the command LIMIT, meaning the query will show a limited result. Such as, we have used LIMIT 3 after closing the WHERE clause, so only three outputs will be in the result section.

<pre> 1 PREFIX woo: <http://infoneer.txstate.edu/ontology/MWOO/> 2 # What are the causes of undeisrable behaviour? 3 SELECT ?ub ?ca 4 WHERE { 5 ?ub rdf:type woo:UndesirableBehavior . 6 ?ub woo:causedBy ?ca 7 } 8 LIMIT 3 </pre>	
<div> <div>Run to File</div> <div>Text</div> <div>Charts</div> <div>Visualize</div> <div>3 Results, 169 ms</div> </div>	
ub	ca
http://infoneer.txstate.edu/ontology/MWOO/WO10-BlewApart	http://infoneer.txstate.edu/ontology/MWOO/WO10-Diffuser
http://infoneer.txstate.edu/ontology/MWOO/WO20-Leaks	http://infoneer.txstate.edu/ontology/MWOO/WO20-LooseFitting
http://infoneer.txstate.edu/ontology/MWOO/WO35-Leaks	http://infoneer.txstate.edu/ontology/MWOO/WO35-Leaking

Figure 36. Query Example 1 with LIMIT Command.

Query Example 2

What are the states caused by undesirable behavior?

This query structure is similar to the first one, but the only difference is that we specifically wanted to know about undesirable behavior causing the state rather than all causes of the state. Hence, we have parameterized state and undesirable beside the SELECT clause and declared their classes in the next two lines in Figure 37. The last line of the query is the duplication of our text query but just in the triple format. The query provided only one result, which is excess vibration is causing the state broken. We also checked the raw workorder text, which is “*exhaust bracket cracked and broken due to excessive vibration*” as well as the graph to validate our findings.

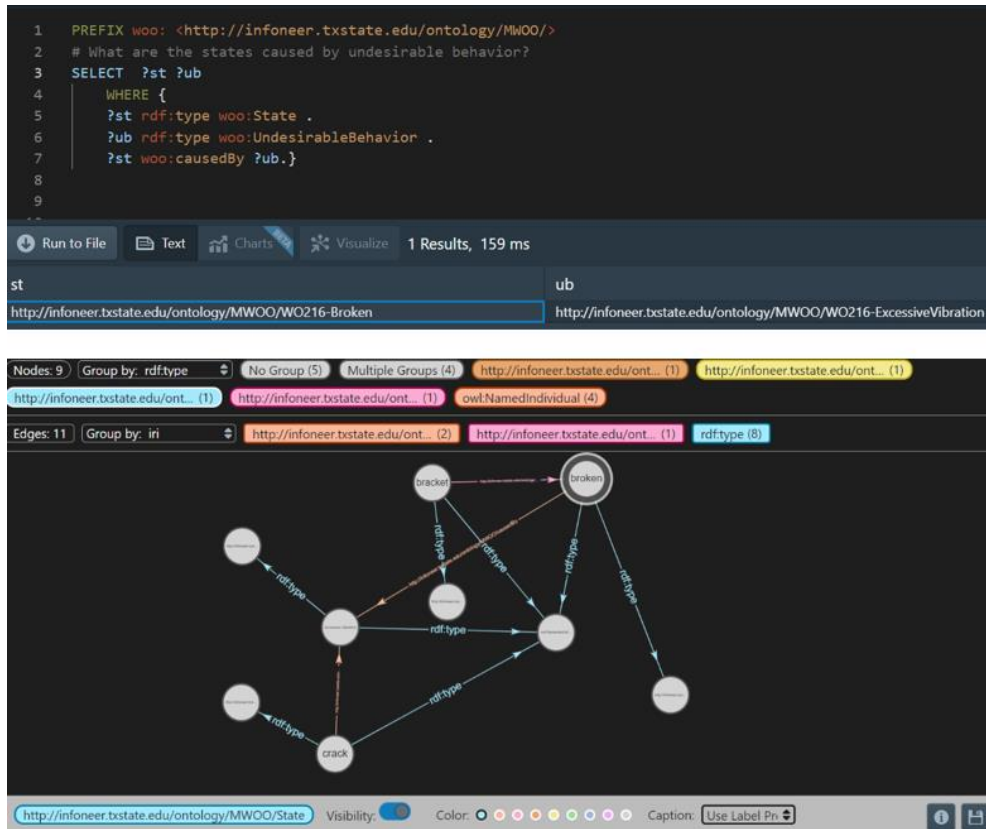


Figure 37. Query Example 2 and Partial Result Visual Presentation.

Query Example 3

What states are caused by Defective Artifacts?

From the ontology, we know that one of the sub-states of the state is **Defunct State** and we want to know if there are both state and defunct state, which are caused by **Defective Artifacts**. The other seven states do not have any instances which is caused by defective artifacts. So, putting those altogether in the query would not affect the result. Figure 38 presents the new 3rd query, where a new command, UNION, has been used to merge the sub-state with the state. The **?st** has been declared as both state and defunct state to accommodate the merging. The query resulted in 9 outputs, and after checking ontology, we confirmed that all nine of them are coming from **Defunct State** class. We chose to visualize maintenance workorder 187, which was “SC C: machine won't drive or

go into gear. C: blown lamp fuse on slot 15 for switch power.” Clearly, the raw text is full of technical jargon, but from the picture illustration, it is clear that a blown fuse is causing the not driving defunct state.

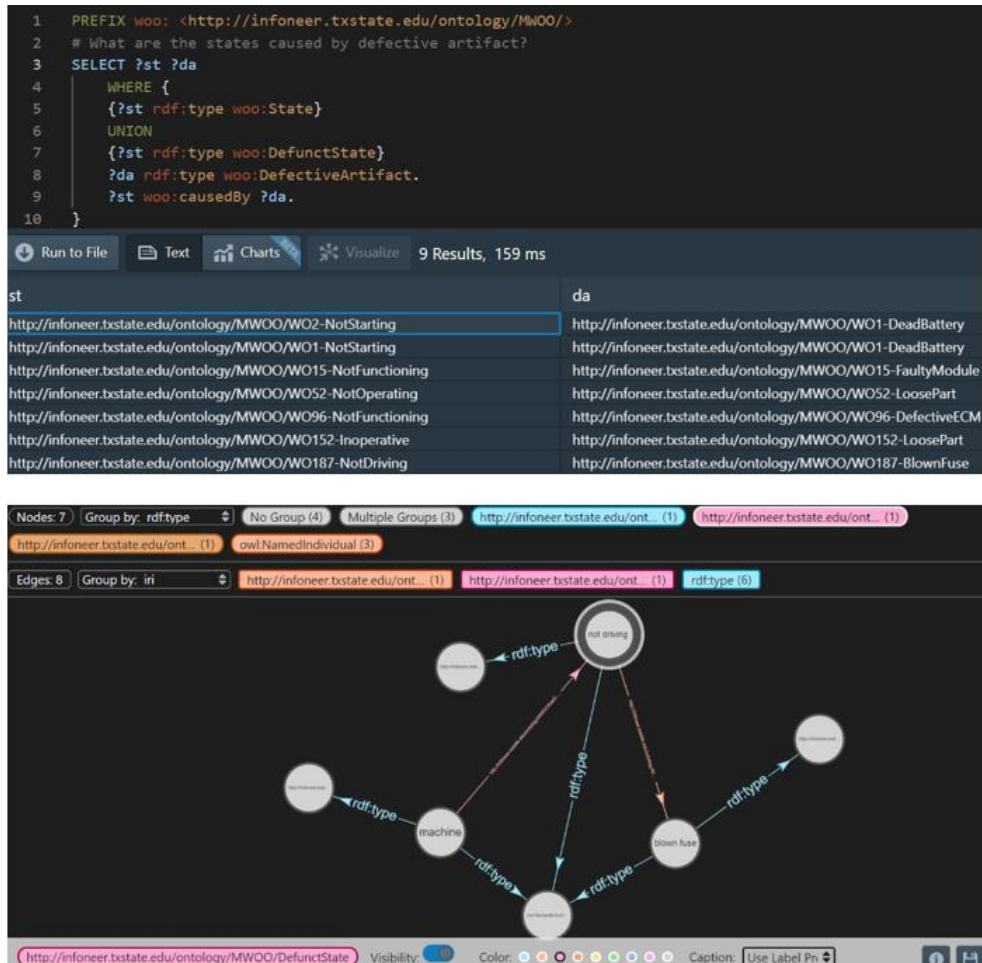


Figure 38. Query Example 3 and Partial Result Visual Presentation.

Query Example 4

What are the reasons for not moving state?

Till now, we only needed to define **rdf:type** to express the belonging under a specific ontology class. However, if we are interested in a specific instance, we will be using **rdfs:label** to refer to that variable shown in Figure 39. In query 4, we were interested to know about the **not moving** state, and we had two results from the query.

We compared the workorder 48 “*machine won't move failed foot switch*” and validated the visualization.

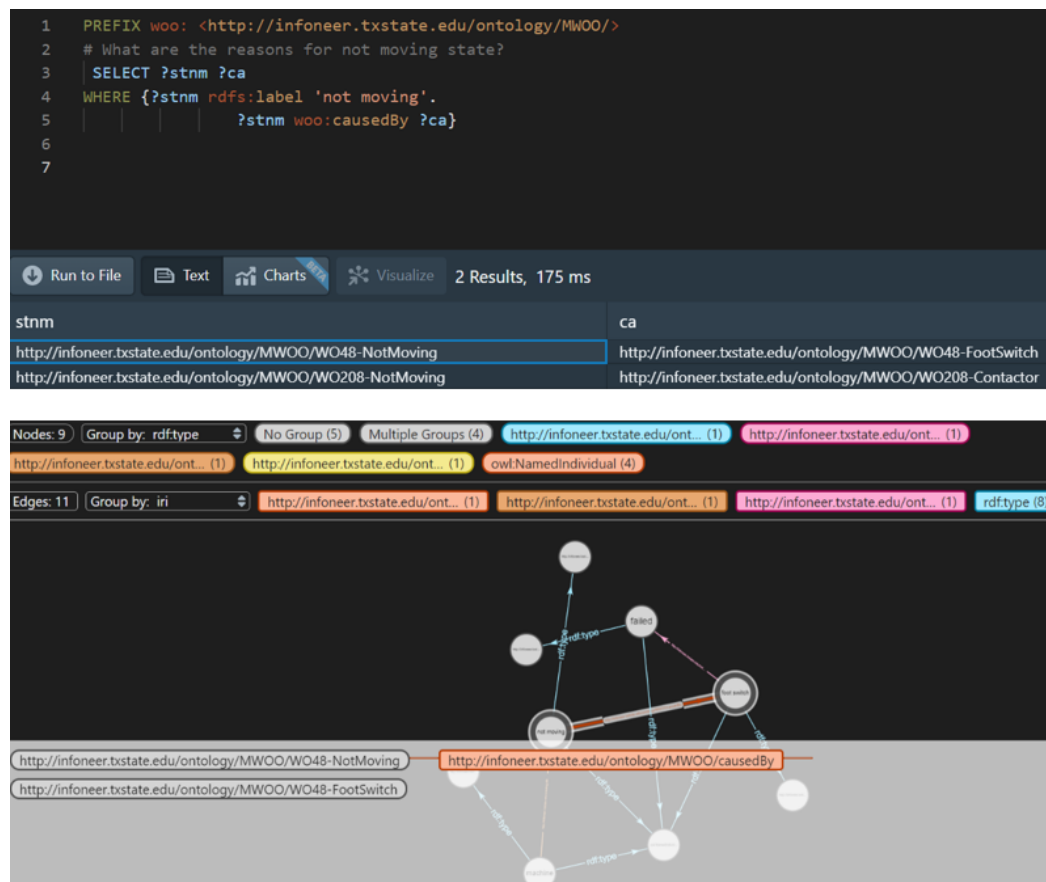


Figure 39. Query Example 4 and Partial Result Visual Presentation.

Query example 5

Is boom a bearer of crack?

This query is different from the previous ones, and the SELECT or WHERE Clause is not used here. When we simply want to know the answer in true and false format, we will use ASK query to do that. Such as, if we want to know whether there is any component in the dataset that is bearer of a defect, the query would be,

ASK {

?a rdfs:type woo:Component.

?b rdf:type woo:Defect.

?a woo:bearerOf ?b.

}

We are simply defining the component and defect parameter within their class and linking those with the object property **bearer of** to verify the result. Depending on this structure we built, ask Stardog if the boom bearer of crack and used the rdfs:label since we want to know about a specific component. The query returned the answer **true**, but without any work order numbers. Hence, to validate if the result was correct or not, we went back to the Ontology and checked all instances of boom under Machine class. Figure 40 reflects that workorder 25 “*jib on unit is loose. Boom to jib pivot bushings found to be cracked and breaking*” is the one with the detected relation, so our query was giving us the right answer. There is no visualization option available for this query since we are asking a question rather than wanting to know any triple relations.

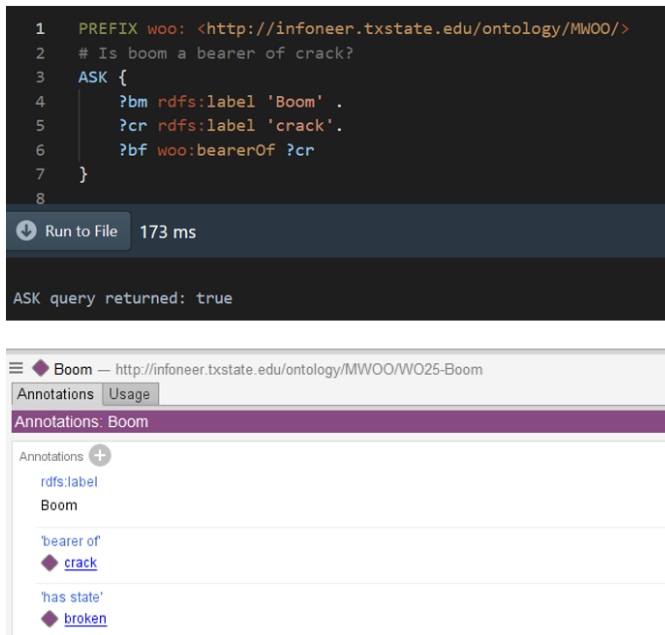


Figure 40. Query Example 5 and Validation from Ontology.

Query Example 6

How many **events** have property **caused by** and if those **caused by** have any **state**?

This query is a bit more complex than the previous ones. We want to know multiple information, so if we simply divide the query into two sections, 1st part is to know the causes of class event but only when the cause occurred due to any state. So, we are targeting to determine two object properties **has state** and **caused by** with our query. We started with parameterized event, cause, and state, eventually by **?e**, **?ca**, and **?st**. Then we the event and state parameters to their respective classes. Finally, we added two additional lines where we want to know the events which are caused by states. The query is shown in Figure 41, and it has provided only one answer, which is failed is caused by a solenoid that is contaminated. It comes from workorder 17, which is “*machine will not move. found brake solenoid failed.*” The two object properties are the edged among solenoid, metal contamination, and failed. One might get confused with the view of the graph, which is much easier to understand compared to the other ones we have shown above. The visibility could be controlled for easy understanding of the semantic relationship, and here, we have only chosen the resultant nodes and edges to be visible.

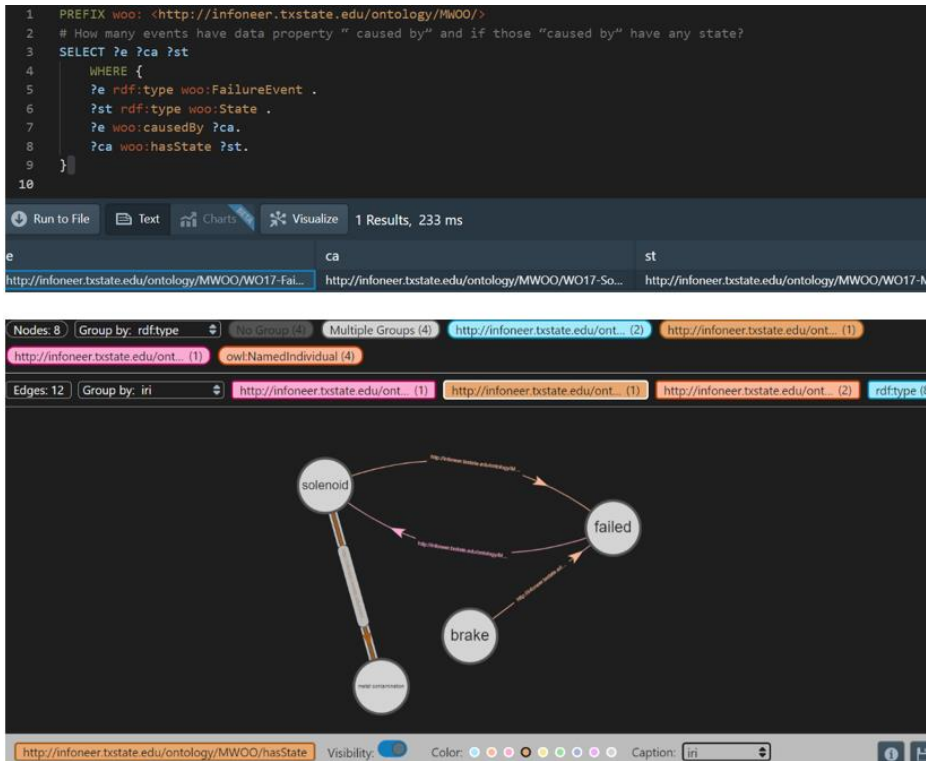


Figure 41. Query Example 6 and Partial Result Visual Presentation.

Query example 7

How many components participates in undesirable behavior and their location is known?

This query structure is similar to query example 6. However, we want to show a side-by-side comparison of changing a simple query structure. Such as, in this query, we want to know the components which participate in undesirable behavior, but their location must be known. We will also show what happens if we just eliminate the condition of known locations; rather, we want components participate in all kinds of undesirable behavior. The alternate query structure is similar to the 2nd query example we presented earlier. Figure 42 illustrates only four undesirable behaviors have known locations out of 46. This also tells us our query structure is correct. We also checked the resulting workorder numbers to see if the query had missed any other location by any chance, but the query was able to detect all the locations successfully.

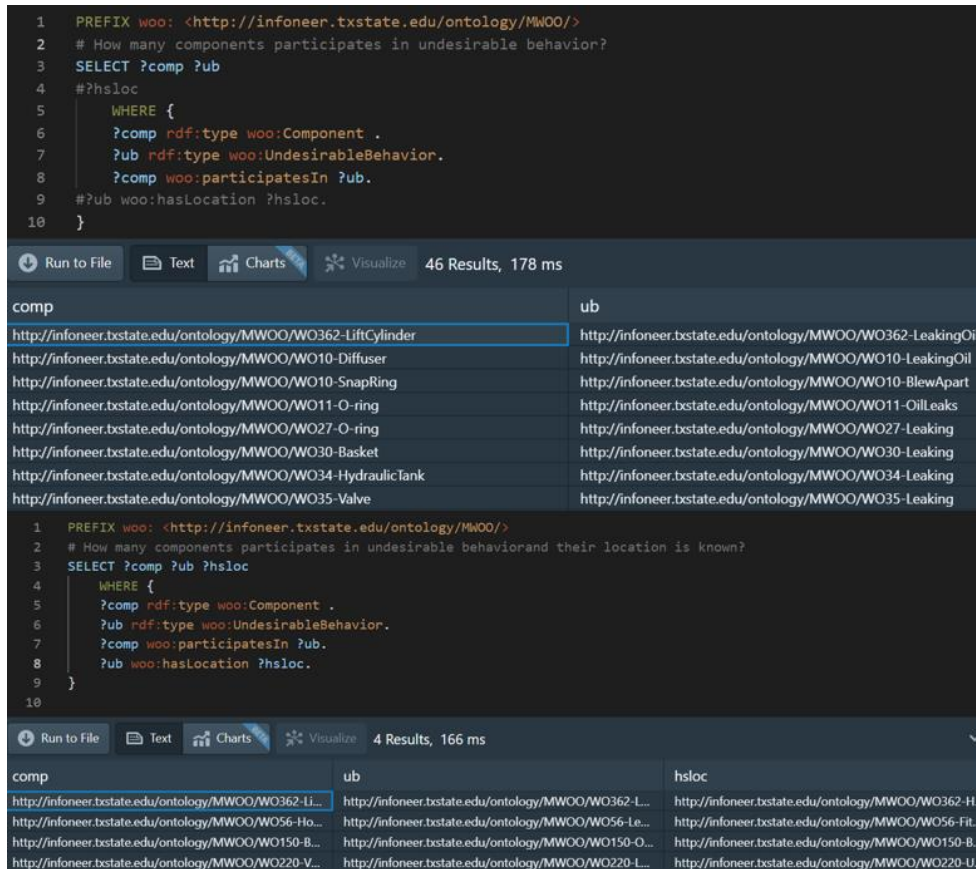


Figure 42. Query Example 7 Comparison.

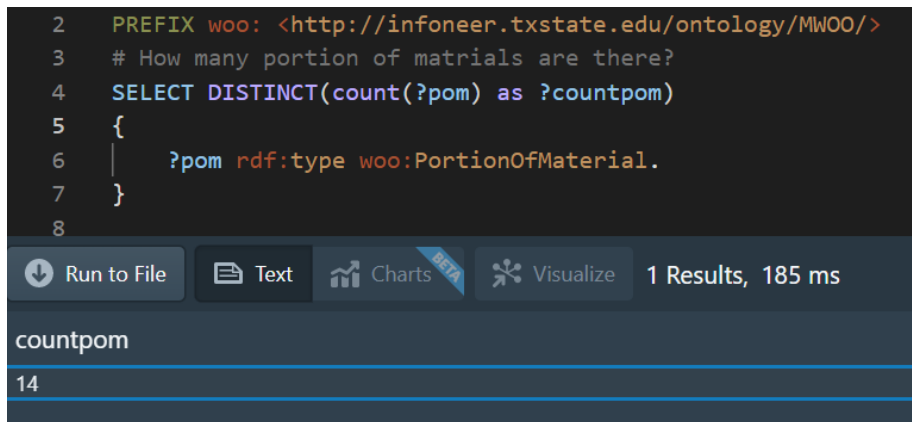
Query Example 8

A. How many portions of materials are there?

B. How many undesirable behaviors per component?

Although our focus here will be on 8B but before that we will introduce COUNT query with the simpler example 8A. The COUNT function counts the number of instances of a variable, and it is used beside the SELECT function. We have also used DISTINCT function no eliminate any repeated instances in the result. Figure 43 picturizes the of portion of material identified by COUNT. The function says that the query wants to count portion of material referred as **?pom**, and the result will be shown with the name **?countpom**. Then, we simply added the class type where the **?pom**

belongs, and we can see there are 14 portion of material present in our knowledge graph database.



```
2 PREFIX woo: <http://infoneer.txstate.edu/ontology/MW00/>
3 # How many portion of materials are there?
4 SELECT DISTINCT(count(?pom) as ?countpom)
5 {
6   ?pom rdf:type woo:PortionOfMaterial.
7 }
8
```

Run to File Text Charts BETA Visualize 1 Results, 185 ms

countpom
14

Figure 43. Query Example 8A.

Now, this brings us to our next query, which is counting the number for each component class instances. In Figure 44, the count function states that we want to know distinct undesirable behavior **?ub** for the component class **?comp**. As a result, we wrote down the class types for the two parameters. But we still need to tell the query what property relationship we have between the component and undesirable behavior to enable the COUNT function to be worked. Participates in is the object property a component could have with undesirable behavior, and that is why we define the relational prefix in the 8th line. The query ends with GROUP BY, which is another new function that will allow grouping the undesirable behavior depending on the instances. Such as in Figure 44, the number of undesirable behaviors for each instance resulted. **FrontAxle** participates in one undesirable behavior, whereas **RunningKeySwitch** has two undesirable behaviors. The undesirable behaviors (edges highlighted in orange) for work orders are also visualized in the simple graphical representation and provide validation of results.

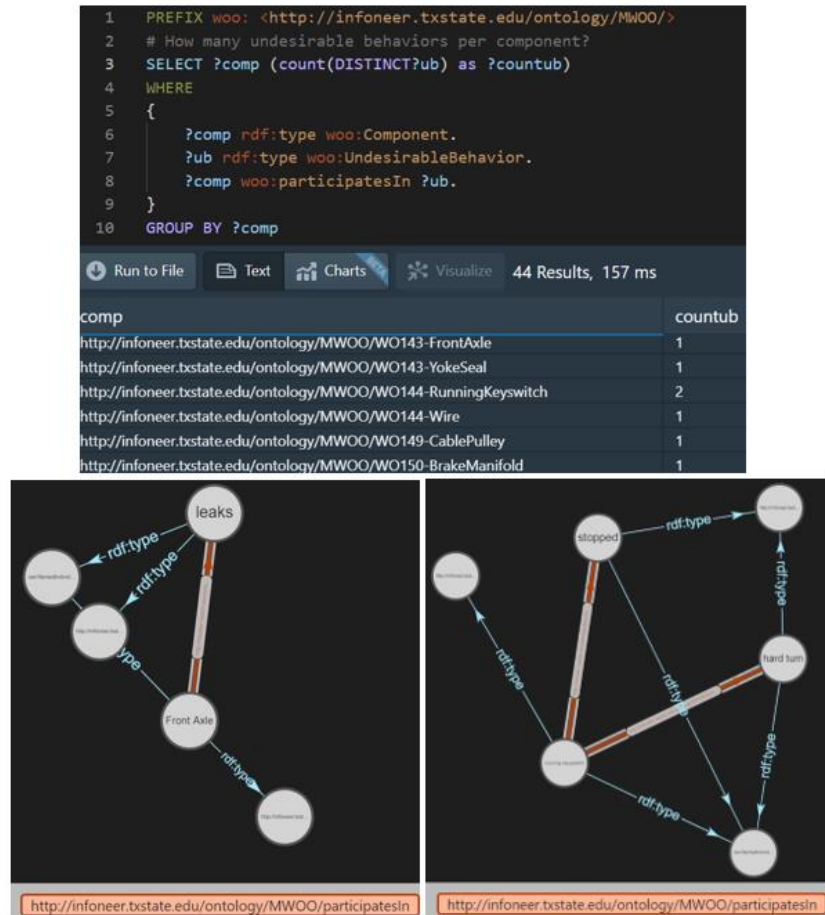


Figure 44. Query Example 8B.

Query Example 9

Find all components excluding seal which participate in leaking, and if that leaking causes a state.

The query question asked for three different pieces of information. First, it is trying to find out only the components which participates in leaking and these components should not include 'seal'. In addition, it also wants to know if these leaking are resulting in any state. This query can be solved using other different functions, but we have chosen to move forward with the straightforward approach shown in Figure 45.

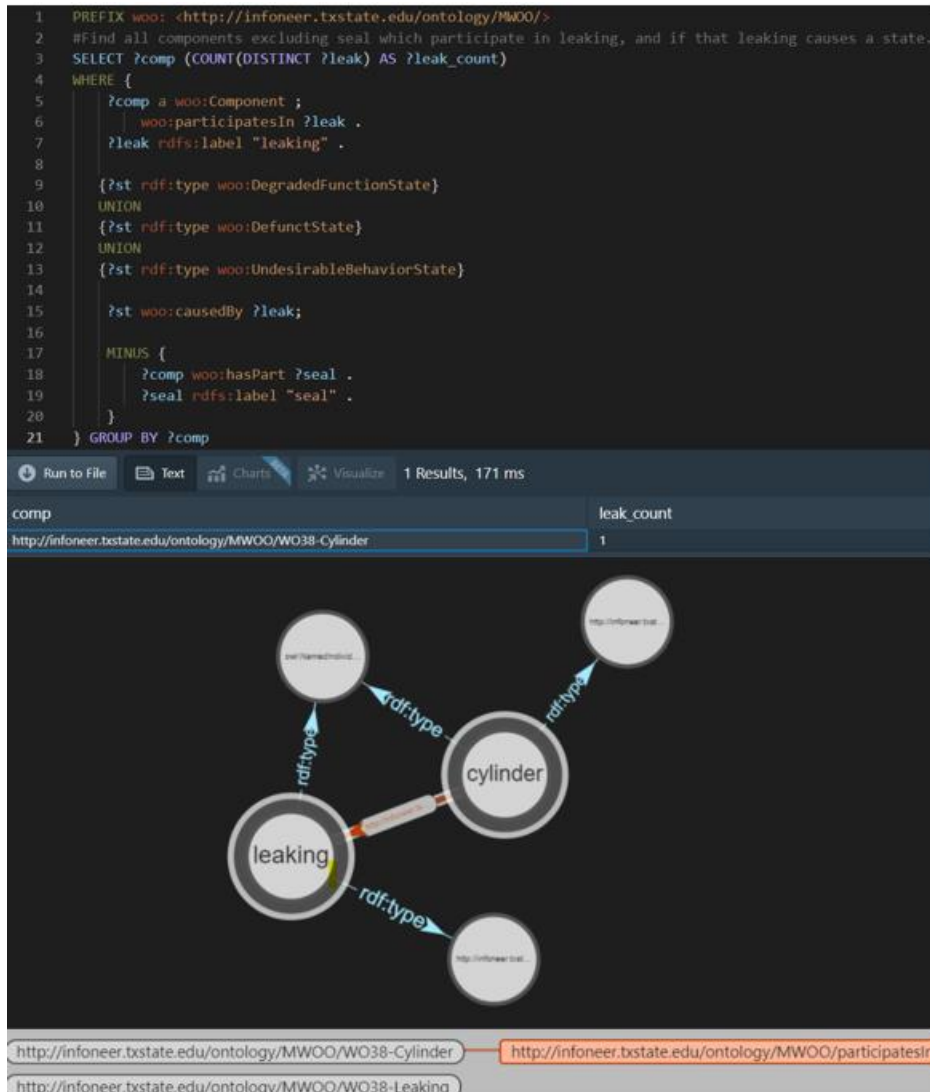


Figure 45. Query Example 9.

We have started with the COUNT function to know the number of leaking for each component followed by the WHERE function. It is visible that we first wanted to know the components that participates in leaking as well as the states the leaking has caused from line 5-15 of the query structure. We used a semi-colon beside component rather than a dot, which allow us to not rewrite the 1st part of the triple structure, and in our case, it is **?comp** for Component. Degraded Function State, Defunct State, and Undesirable Behavior State are the subset of **State** in our OWL ontology. Hence, we have

used the UNION function to structure that. Subsequently, we used a new function called MINUS to exclude the seal from the list of components and ended with GROUP BY function to get the result categorized for each component. The query resulted in only one result WO 38, and we have validated the result by going back to our raw text “ *Brakes not working brake cylinder leaking down.*” The not working is the only state leaking is causing, and the visualization shows that the component cylinder participates in leaking.

Validation Using Reasoner

Protégé tool has the plug-in called **Reasoner** and by running this plug-in, we can generate the inferred relationship between two instances of the ontology classes. When we are generating knowledge graphs to show meaningful relationships among instances, we called these relationships **Asserted** type. Whereas, if our ultimate goal is to build a semantic model, meaning both human and machine will have equal understanding understand of the model and they will have a common ground of communicating with each other, then the next step should be utilizing the Reasoner. It creates the inferred relationship among instances and allow better understanding of how one instance is connected to other. Figure 46 is an example of our ontology without using the Reasoner function and how the asserted relationships look like. The **Unit With Leak Failure** is an instance of **Nonconforming Artifact** and the definition of the instance is provided as a natural language definition. This definition is for the use of human model developer for creating meaningful triple relations while generating the knowledge graphs. However, Protégé does not understand this definition and requires an equivalent definition so that by hitting the Reasoner plug-in, it can generate indirect relationships.

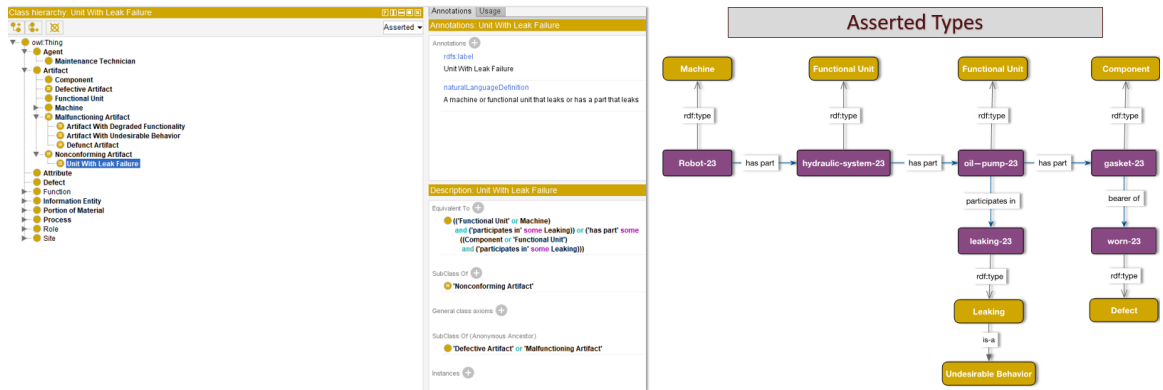


Figure 46. Asserted Relations without Reasoner.

The right side of the image is an example of how the asserted type of relation looks like without using the reasoner. We can see that robot has part hydraulic system which also has a part called oil pump. Besides, gasket is also a part of the oil pump. But previously shown in Chapter 3, Figure 24, we talked about how the knowledge graph could be extended and, in the description, we mentioned that if machine has part functional unit and functional unit has part component, then the inferred relationship is that the component is also a part of the machine. Same for this example, we can see that the asserted relationships are robot has part hydraulic system, hydraulic system has part oil pump and oil pump has part gasket. It also means that both oil pump and gasket are part of the robot as well as gasket is also a part of hydraulic systems. The other inferred relation is if oil pump participates in leaking, then both robot and hydraulic systems are type of unit with leak failure. Furthermore, if gasket is a bearer of defect then robot is type of defective artifact too. The inferred relations can be seen in Figure 47.

If we put the equivalent definition of the natural language definition under **Equivalent To** correctly and run the reasoner, the instances among which the inferred relationship works, would pop up under the instances. As example, in figure 46, we did not have any instances for unit with leak failure but after running the reasoner we can see

automatically generated instances- hydraulic systems, oil pump and robot. It validates that we have correctly provided the equivalent definition and one can easily understand the difference between asserted and inferred relations. The instance background will turn yellow, and the user can also go back and forth between asserted and inferred from the drop-down menu. Hence, we used this plug-in to analyze if our reasoning is correct and if the inferred relations could be generated successfully.

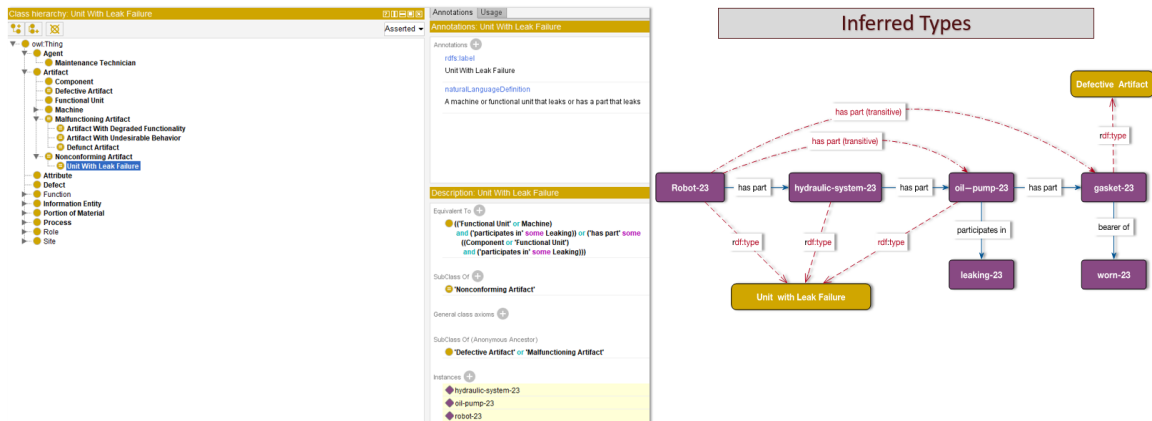


Figure 47. Inferred Relations with Reasoner.

IV. CONCLUSION

In this chapter, we will discuss how our implemented methodology has satisfied the research questions we have identified in Chapter 1.

What main concept categories and sub-categories can be used for classifying the key terms that appear in MWO data? What should be the top-level concepts of the MD thesaurus?

When we started working with the raw text data, the most appeared terms were different machine and component types, the failures they cause and how those affect other individuals, the location of parts and the failures, as well as the environmental condition. As a result, we classified and sub-classified them to create a standardized framework that can be used not only for our dataset but also for any maintenance dataset. The main class concepts were the top concepts, and they are: **Artifacts, Condition, Event, Function, Maintenance Treatment, Material Substances** and **Property**. The thesaurus and ontology are both extendable, and we have earlier visualized the main concept classes and sub-classes in Figure 2 and Figure 3 and provided their definition in Table 5.

Does using tools with NLP support (such as Nestor) improve the efficiency of the tagging process?

The answer is both yes and no. There is a lot of noise in raw data and tools such as Nestor can be helpful by removing noise factors. In our case, Nestor was able to detect the required taggable words filtering the noise as well as identifies both single and multi-phrase words together. However, the tool is newly created and still needs appropriate updates to work faster. Since we were able to use the primitive version of the tool, it was

very time consuming to tag our larger number of datasets. Each time we uploaded only ten raw work orders to achieve a faster result out of five hundred total maintenance raw data. Also, we got access to the tool in the middle of our research work, so by then, we already had tagged most of the required terms of our dataset by utilizing the **Entity Extractor** functionality of the SKOS tool. Nestor has been used mostly for our tagging check to identify any of the important terms we missed during manual tagging.

[How to categorize the Nestor-tagged problems under the right category/bucket in the thesaurus?](#)

As previously mentioned, Nestor is a generalized NLP tool and mainly identifies the **Items**, **Problems**, **Solution**, **Problem Items**, and **Ambiguous** terms in the raw text. Adding the solution was not the scope of this research, and the ambiguous terms were subject to our manual exploration if they would fall under any concept bucket of the thesaurus.

- The **Items** were mapped under the **Asset**, **Component**, and **Functional Unit**.
- **Problems** were mapped under **Nonconforming Condition** and **Failure Event**.
- **Problem Items** were also mapped under **Nonconforming Condition**.

[How effective our final model will be in showing the semantic relationship among various entities?](#)

The final model was extremely effective in showing the semantic relationship among the entities. We started with the raw text and were able to cluster the terms in the thesaurus, which in the end, resulted in the knowledge graph. When we have a set of standardized concept sets, it also impacts our mental ability to recognize the list of problems in the maintenance workorders easily. It allows a better understanding of the

root cause of the problem when visualizing the knowledge graphs and what relationship one instance possesses with others in the same work order. Especially, the workorders consisting of a lot of issues were often cumbersome to spot the root cause due to the technical jargon, and these graphical representations were a big aid to meet that need. We have presented examples from both the RDF Grapher online visualization tool and the Stardog query platform to illustrate the internal semantic relations.

How would we express the queries to satisfy our competency questions, such as:
What is the cause of this maintenance problem, where is the location of the problem,
what are the WOs related to this problem (s), and so on.

The SPARQL structure is fixed, and any knowledge graph platform could be used to generate the SPARQL queries. We have followed the Stardog platform query tutorials to understand how we can generate meaningful queries which would retrieve whatever information we want to know. We have provided nine query examples using our knowledge graphs as required input and the queries were successfully able to show the maintenance problems, location along with their work order numbers. Such as, our query retrieved the **Artifacts** along with their **Workorders** that participates in the undesirable behavior as well as the **Undesirable Behavior** results presented in Query Example 1 and 2. In addition, Query example 7 retrieved the known location for the **Undesirable Behavior** of the **Artifacts**.

One other advantage of using the SPARQL was to be able to determine the true percentage of the retrieved result. We are dealing with a larger dataset, and the number of instances under the concept classes in our ontology was approximately 700 from the two hundred and fifty workorders presented in Figure 48. The SPARQL returns the count of

individuals under all classes, and we have put them on a histogram after sorting them from largest to smallest. Since the individual count is pretty big, duplication of the result is possible as well as getting a larger output. In those cases, we have effectively used the **DISTINCT** and **LIMIT** functions to get rid of any duplication or limit the output number.

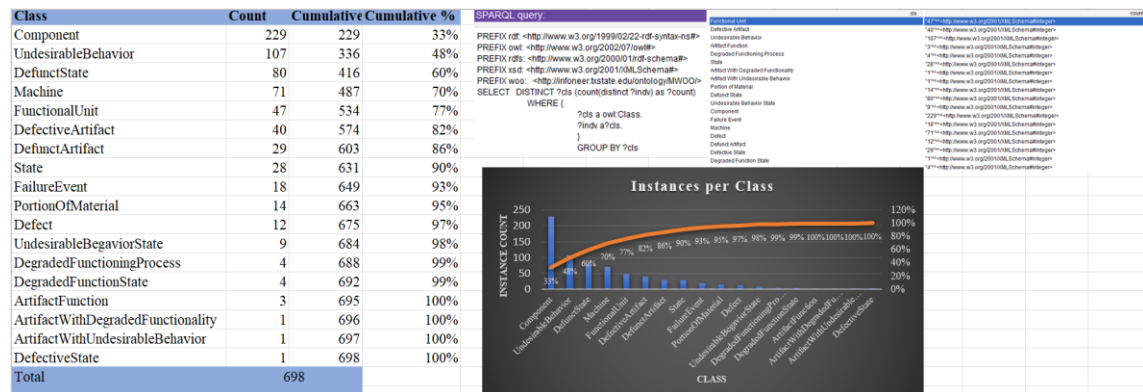


Figure 48. Instance Count per Class in Ontology.

However, we did our own sanity check by every time getting back to the ontology and raw text data to validate the true percentage, meaning if a query is showing ten **Defunct State**, is it true that all of them are **Defunct State**. This is called the precision and recall performance metrics we talked about as a part of our Task 4.

Getting back to our raw maintenance text data, we have seen in our provided examples that the raw texts are full of technical jargon, and especially if somebody is not from the manufacturing or maintenance field, it is a struggle for them to find out the meaning. Having thesaurus followed by Ontology allows data all the data collection schema to use this shared vocabulary. Knowledge graphs also have these technical terms in a graphical form but without any jargon, stop words, unnecessary punctuation, and ambiguous words. The concepts and relationships that are allowed in knowledge graphs are defined by ontology. The structure of a knowledge graph facilitates the integration of several heterogeneous data sets from various companies of varying quality. With each

item or instance represented only once, together with all of its relationships, in the context of all of the other subjects and their relationships, Knowledge Graphs offer a model of how everything is related. This enables one to understand how everything is connected on a broad scale. They enable users to quickly conduct exploratory queries against massive amounts of data without creating indexes or otherwise tailoring datasets for particular queries. Figure 49 is a side-by-side comparison of raw text and a knowledge graph.

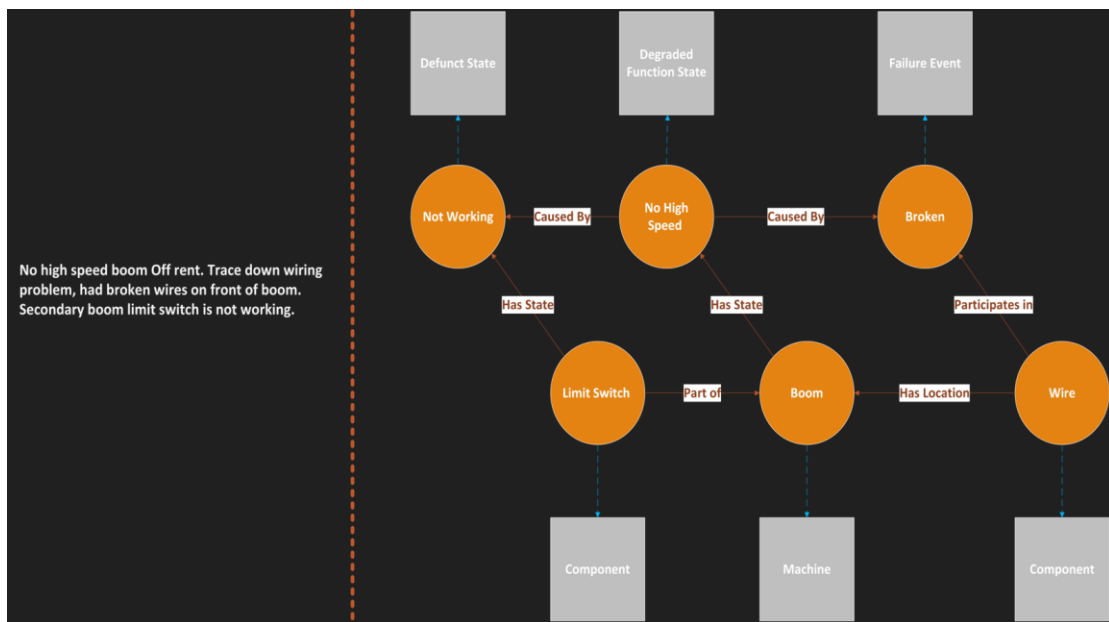


Figure 49. Raw Text and Knowledge Graph Comparison.

Clearly, the raw text consists of a lot of smaller issues such as **Broken Wire**, **Wiring problem**, **No high speed**, and unnecessary terms like **Off rent**. Lack of punctuation and grammatical structure is making the text hard to understand and is the difficult root cause of the problem. Whereas the knowledge graph on the right side is not only expressing explicit meaning but also their exhibiting the relationships between each individual. One can easily understand that the bigger failure is that the boom is running at a low speed which is caused by both broken wire and the nonfunctioning limit switch.

Besides, the broken wire is located in boom, and the limit switch is a part of the boom. Without converting the texts into a graph, one can hardly imagine how many hidden meanings a raw text can convey. In our case, each raw text of the five hundred total raw texts possesses useful information and that can lead to fruitful failure cause mapping. Furthermore, it is not possible to conduct such a structured breakdown of the problem only from the raw text. Raw text is the storehouse of our knowledge graph where we can reflect back to build the graphs piece by piece. However, when we are progressing fast toward Industry 4.0, we definitely need to have smart failure detection, and the knowledge graph indeed accomplishes that need.

The first future work of this research work includes adding more work orders to the master triples file and to examine how Stardog or our internally developed query tool can perform query and retrieve accurate data. Currently we have utilized two hundred and fifty workorders to generate the knowledge graph. Other half of the workorders could be loaded to generate the knowledge graphs for entire workorder dataset and then conduct the query to validate the date retrieving accuracy. The second future work would be adding the solution of each work order. This research work has only focused on analyzing and tagging the problem statement of related to a work order. Nevertheless, adding the solution would be the next approach as the ultimate goal is to find a solution to a maintenance breakdown or failure as soon as possible and use the knowledge graph as a medium of preventive measure.

Finally, generating SWRL rules to expand the knowledge graph is one possible direction for future work. SWRL allows automatically adding new triples to the graph by combining two entities of indirect relationships. Such as considering a concept called

State in the Owl ontology. The state directly relates to the **Event**, another concept of the ontology, and the instances of the Event, such as failed or ruptured, could be caused by a defective artifact. Although the state and the defective artifact do not directly connect with the SWRL approach, we can generate the indirect triple relationship visible in the knowledge graph. Using SWRL will search the entire graph and add this new triple whenever a similar pattern appears, which will result in a lot of new triples. It will ensure a greater root cause analysis of the maintenance work orders' problems.

REFERENCES

- Ameri, F., & Yoder, R. (2019). A Thesaurus-Guided Method for Smart Manufacturing Diagnostics. IFIP International Conference on Advances in Production Management Systems, 722–729.
- Ameri, F., Yoder, R., & Zandbiglari, K. (2020). SKOS Tool: A Tool for Creating Knowledge Graphs to Support Semantic Text Classification. In B. Lalic, V. Majstorovic, U. Marjanovic, G. von Cieminski, & D. Romero (Eds.), *Advances in Production Management Systems. Towards Smart and Digital Manufacturing* (pp. 263–271). Springer International Publishing. https://doi.org/10.1007/978-3-030-57997-5_31
- Arif-Uz-Zaman, K., Cholette, M. E., Ma, L., & Karim, A. (2017). Extracting failure time data from industrial maintenance records using text mining. *Advanced Engineering Informatics*, 33, 388–396.
- Bokinsky, H., McKenzie, A., Bayoumi, A., McCaslin, R., Patterson, A., Matthews, M., Schmidley, J., & Eisner, L. (2013). Application of natural language processing techniques to marine V-22 maintenance data for populating a CBM-oriented database. 463–472.
- Brundage, M. P., Sexton, T., Hodkiewicz, M., Dima, A., & Lukens, S. (2021). Technical language processing: Unlocking maintenance knowledge. *Manufacturing Letters*, 27, 42–46.
- Devaney, M., Ram, A., Qiu, H., & Lee, J. (2005). Preventing failures by mining maintenance logs with case-based reasoning.

- Dima, A., Lukens, S., Hodkiewicz, M., Sexton, T., & Brundage, M. P. (2021). Adapting natural language processing for technical text. *Applied AI Letters*, 2(3), e33.
<https://doi.org/10.1002/ail2.33>
- Gao, Y., Woods, C., Liu, W., French, T., & Hodkiewicz, M. (2020). Pipeline for Machine Reading of Unstructured Maintenance Work Order Records (p. 1408).
https://doi.org/10.3850/978-981-14-8593-0_3888-cd
- Gharehchopogh, F. S., & Khalifelu, Z. A. (2011). Analysis and evaluation of unstructured data: Text mining versus natural language processing. 2011 5th International Conference on Application of Information and Communication Technologies (AICT), 1–4.
- Gunay, H. B., Shen, W., & Yang, C. (2019). Text-mining building maintenance work orders for component fault frequency. *Building Research & Information*, 47(5), 518–533. <https://doi.org/10.1080/09613218.2018.1459004>
- Hodkiewicz, M., & Ho, M. T.-W. (2016). Cleaning historical maintenance work order data for reliability analysis. *Journal of Quality in Maintenance Engineering*, 22(2), 146–163. <https://doi.org/10.1108/JQME-04-2015-0013>
- Hodkiewicz, M., Lukens, S., Brundage, M. P., & Sexton, T. (2021). Rethinking Maintenance Terminology for an Industry 4.0 Future. *International Journal of Prognostics and Health Management*, 12(1), Article 1.
<https://doi.org/10.36001/ijphm.2021.v12i1.2932>

Hossayni, H., Khan, I., Aazam, M., Taleghani-Isfahani, A., & Crespi, N. (2020).

SemKoRe: Improving Machine Maintenance in Industrial IoT with Semantic

Knowledge Graphs. *Applied Sciences*, 10(18), 6325.

<https://doi.org/10.3390/app10186325>

Learn OWL and RDFS. (n.d.). Cambridge Semantics. Retrieved October 11, 2022, from

<https://cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/>

Learn RDF. (n.d.). Cambridge Semantics. Retrieved October 11, 2022, from

<https://cambridgesemantics.com/blog/semantic-university/learn-rdf/>

McKenzie, A., Matthews, M., Goodman, N., & Bayoumi, A. (2010). Information

Extraction from Helicopter Maintenance Records as a Springboard for the Future

of Maintenance Text Analysis. In N. García-Pedrajas, F. Herrera, C. Fyfe, J. M.

Benítez, & M. Ali (Eds.), *Trends in Applied Intelligent Systems* (pp. 590–600).

Springer. https://doi.org/10.1007/978-3-642-13022-9_59

Mohan, V. (2015). Preprocessing Techniques for Text Mining—An Overview.

[https://www.researchgate.net/profile/Vijayarani-](https://www.researchgate.net/profile/Vijayarani-Mohan/publication/339529230_Preprocessing_Techniques_for_Text_Mining_-_An_Overview/links/5e57a0f7299bf1bdb83e7505/Preprocessing-Techniques-for-Text-Mining-An-Overview.pdf)

[Mohan/publication/339529230_Preprocessing_Techniques_for_Text_Mining -](https://www.researchgate.net/profile/Vijayarani-Mohan/publication/339529230_Preprocessing_Techniques_for_Text_Mining_-_An_Overview/links/5e57a0f7299bf1bdb83e7505/Preprocessing-Techniques-for-Text-Mining-An-Overview.pdf)

[An_Overview/links/5e57a0f7299bf1bdb83e7505/Preprocessing-Techniques-for-](https://www.researchgate.net/profile/Vijayarani-Mohan/publication/339529230_Preprocessing_Techniques_for_Text_Mining_-_An_Overview/links/5e57a0f7299bf1bdb83e7505/Preprocessing-Techniques-for-Text-Mining-An-Overview.pdf)

[Text-Mining-An-Overview.pdf](https://www.researchgate.net/profile/Vijayarani-Mohan/publication/339529230_Preprocessing_Techniques_for_Text_Mining_-_An_Overview/links/5e57a0f7299bf1bdb83e7505/Preprocessing-Techniques-for-Text-Mining-An-Overview.pdf)

Navinchandran, M., Sharp, M. E., Brundage, M. P., & Sexton, T. B. (2021). Discovering

critical KPI factors from natural language in maintenance work orders. *Journal of*

Intelligent Manufacturing. <https://doi.org/10.1007/s10845-021-01772-5>

Nestor. (2021). [Python]. National Institute of Standards and Technology.

<https://github.com/usnistgov/nestor> (Original work published 2020)

OWL Web Ontology Language Overview. (n.d.). Retrieved October 11, 2022, from
<http://www.ksl.stanford.edu/people/dlm/webont/OWLOverviewMay12003.htm>

Owl 101. (n.d.). Cambridge Semantics. Retrieved October 11, 2022, from
<https://cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/owl-101/>

RDF - Semantic Web Standards. (n.d.). Retrieved October 11, 2022, from
<https://www.w3.org/RDF/>

Ringsquandl, M., Kharlamov, E., Stepanova, D., Lamparter, S., Lepratti, R., Horrocks, I., & Kröger, P. (2017). On event-driven knowledge graph completion in digital factories. 2017 IEEE International Conference on Big Data (Big Data), 1676–1681. <https://doi.org/10.1109/BigData.2017.8258105>

Semantic Web—W3C. (n.d.). Retrieved October 11, 2022, from
<https://www.w3.org/standards/semanticweb/>

Sexton, T., Brundage, M. P., Hoffman, M., & Morris, K. C. (2017). Hybrid datafication of maintenance logs from ai-assisted human tags. 2017 Ieee International Conference on Big Data (Big Data), 1769–1777.

Sexton, T. B., Brundage, M. P., Hodkiewicz, M., & Smoker, T. (2018). Benchmarking for keyword extraction methodologies in maintenance work orders.
<https://www.nist.gov/publications/benchmarking-keyword-extraction-methodologies-maintenance-work-orders>

Sexton, T. B., & Brundage, M. B. (2019). Nestor: A Tool for Natural Language Annotation of Short Texts. Journal of Research of the National Institute of Standards and Technology, 124, 124029. <https://doi.org/10.6028/jres.124.029>

- SKOS Simple Knowledge Organization System Primer. (n.d.). Retrieved October 11, 2022, from <https://www.w3.org/TR/2009/NOTE-skos-primer-20090818/>
- SKOS Simple Knowledge Organization System—Home page. (n.d.). Retrieved October 11, 2022, from <https://www.w3.org/2004/02/skos/>
- Spasic, I., Ananiadou, S., McNaught, J., & Kumar, A. (2005). Text mining and ontologies in biomedicine: Making sense of raw text. *Briefings in Bioinformatics*, 6(3), 239–251. <https://doi.org/10.1093/bib/6.3.239>
- Qiao, B., Fang, K., Chen, Y., & Zhu, X. (2017). Building thesaurus-based knowledge graph based on schema layer. *Cluster Computing*, 20(1), 81–91. <https://doi.org/10.1007/s10586-016-0725-z>
- Valdez, J., Rueschman, M., Kim, M., Redline, S., & Sahoo, S. S. (2016). An Ontology-Enabled Natural Language Processing Pipeline for Provenance Metadata Extraction from Biomedical Text (Short Paper). *On the Move to Meaningful Internet Systems ... : CoopIS, DOA, and ODBASE : Confederated International Conferences, CoopIS, DOA, and ODBASE ... Proceedings. OTM Confederated International Conferences*, 10033, 699–708. https://doi.org/10.1007/978-3-319-48472-3_43
- What is RDFS? (2021, July 25). <https://www.bobdc.com/blog/whatisrdfs>
- What is RDF? (2021, June 27). <https://www.bobdc.com/blog/whatisrdf/>