IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 1 June 2024, accepted 28 July 2024, date of publication 12 August 2024, date of current version 10 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3441757

RESEARCH ARTICLE

Semantic Quality Assurance of Industrial Maintenance Procedures

CAITLIN WOODS[1], MELINDA HODKIEWICZ[2], (Member, IEEE), AND TIM FRENCH[1]
[1]Department of Computer Science and Software Engineering, The University of Western Australia, Crawley, WA 6009, Australia
[2]School of Engineering, The University of Western Australia, Crawley, WA 6009, Australia

Corresponding author: Caitlin Woods (caitlin.woods@uwa.edu.au)

This work was supported by Australian Government Research Training Program (RTP) Scholarship.

ABSTRACT Maintenance technicians in industry follow procedures that guide them through inspection, repair, and service tasks. Organisations seek to convert procedure documentation to machine-readable formats as their digital capabilities improve and regulatory requirements tighten. In this paper, we consider the opportunity for semantic quality assurance of digital procedures. We demonstrate a configurable and repeatable workflow containing three modules. The *completeness module* makes implicit information in procedures explicit using OpenAI's Generative Pre-trained Transformer (GPT) model. The *consistency module* creates Resource Description Framework (RDF) triples that are aligned with, and checked against, the axioms of the open-source Ontology for Maintenance Procedure Documentation (OMPD). Finally, the *correctness module* performs closed-world checks on the RDF triples using the Shapes Constraints Language (SHACL). Each module can be used in isolation, or together, to realise an end-to-end semi-automated quality assurance workflow. Pre-processing of the raw maintenance procedure documents to extract entities (tools, materials and activities) and relations is achieved in a novel manner using prompt engineering with OpenAI's GPT-3.5 Turbo model and few-shot learning. This end-to-end workflow enables organisations to perform quality assurance such as assessing the correct order for task sequences, and checking that all maintenance procedures have at least one maintenance task. We demonstrate this workflow on six procedures from the iFixit repository. The outputs of this workflow support maintenance technicians, planners and engineers by realising high-quality procedure documentation and automated procedure management update processes. The code and data used in this work is publicly available at https://github.com/equonto/quokka/.

INDEX TERMS Industrial ontology, ontology templates, OpenAI GPT, OTTR, SHACL, technical language processing.

I. INTRODUCTION

Maintenance procedures are a crucial part of a maintenance technician's safe work practices. Maintenance work can be planned (i.e. inspections, services and equipment overhauls) or unplanned (i.e. repair following equipment failure). Transforming procedure documentation to digital formats is a topic of interest for many industrial organisations. The opportunity in this digital transformation is to create machine-readable artifacts. Machine-readable, semantic maintenance procedures using the Resource Description Framework (RDF) and the Web Ontology Language (OWL) enable data quality improvement via reasoning and constraint checking.

In this paper, we demonstrate the use of an open-source Ontology for Maintenance Procedure Documentation (OMPD), along with a semi-automated workflow, to transform maintenance procedure documentation into a machine-readable format for data quality checking. Our workflow, titled *Quokka*[1], is implemented as a Python-based console application. Our implementation is publicly under a Creative Commons License[2] at https://github.com/equonto/quokka/. The purpose of Quokka is to provide support for organisations wishing to implement machine-readable maintenance

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

[1]A Quokka is a marsupial commonly found on the offshore islands of Western Australia. We chose this name for our workflow to align with the naming conventions of the tools from our research group (the UWA NLP TLP Group), where we name tools after Australian animals.

[2]CC BY-NC-SA 3.0 DEED: https://creativecommons.org/licenses/by-nc-sa/3.0/deed.en

procedures. Quokka achieves this by supporting technical writers (i.e. maintenance engineers) in improving the *completeness*, *consistency* and *correctness* of maintenance procedures that conform to the OMPD data model. We improve the *completeness* of the dataset by extracting intrinsic knowledge from the unstructured text in procedures. We check the *consistency* of the data by mapping it to an OWL ontology for reasoning. Finally, we check the *correctness* of the data according to a set of Shapes Constraint Language (SHACL) shapes. The novelty of this paper is the cumulative effects of its parts. We use a combination of Semantic Web Technologies (SWTs) (i.e. information extraction, ontology templates, OWL reasoning and SHACL shapes) to create a technical solution to a real-world data quality use case. The original work was done on real industry maintenance procedures. These cannot be published due to commercial in confidence considerations. As a result of this constraint we demonstrate the workflow using the maintenance procedures publicly available on the iFixit website.[3] Specifically, we use six procedures from the MyFixit dataset [1] - a preprocessed set of iFixit procedures available under a Creative Commons Licence.[4]

In Section II of this paper, we discuss how maintenance procedures fit into a maintenance technician and maintenance engineer's work system and recent works in semantic quality assurance for industrial applications. Section III describes OMPD, the target data model for this work. In Section IV we give give an of the Quokka workflow and describe our Python-based implementation as three modules (completion module, consistency module and correctness module) in Section V. We demonstrate the Quokka workflow using six procedures in Section VI. Finally, in Section VII, we detail our contributions, and reflect on remaining opportunities for future research.

II. BACKGROUND

A. PROCEDURE DOCUMENTATION IN MAINTENANCE

In industry, maintenance procedures are historically developed by maintenance engineers and maintenance planners using Microsoft Word or Excel and stored in PDF or related formats in the Computerised Maintenance Management System (CMMS). For maintenance technicians, common practice involves finding a procedure document for a work order in the CMMS, downloading and printing the document for use during the task, and sending the hand-annotated document back to the maintenance planner once the task is complete [2]. There is increasing use of digital tablets and phones in the workplace. Many companies are transitioning away from paper and read-only formats to digital formats that enable data capture by the field technicians using their portable devices. This transition to digital maintenance procedures raises several opportunities for maintenance technicians. For example, the ability to use different presentation formats (i.e. text, photos, and video), and suggest updates and improvements about to procedures while on the job [3]. This digital transition is also a challenge for the maintenance engineers responsible for creating and updating procedures. In many organisations procedures have been developed over the years with some inconsistency in vocabulary, syntax and format. This transition is an opportunity to standardise maintenance procedures and implement quality control processes. However to do this manually, procedure by procedure, is very costly and a degree of automation is required.

B. SEMANTIC QUALITY ASSURANCE FOR INDUSTRIAL APPLICATIONS

The development of Transformer Language Models (TLM), Large Language Models (LLM) and the release in November 2022 of OpenAI's ChatGPT[5] has fundamentally changed expectations about how structured and unstructured text held in organisational databases can be located, manipulated and queried. The opportunities provided by these new tools and models is driving changes in enterprise architectures and a growing interest in knowledge graphs to augment traditional relational database structures. As the desire by management to query data across different databases increases, semantic interoperability is emerging as a key barrier. To manage this organisations are turning to the semantic web technologies (SWT) that underpin the design of the World Wide Web and are documented by the W3C standards.[6] Semantic languages such as RDF and OWL are gaining popularity, for example in the IoT domain [4], [5] and some organisations now consider RDF as a ''protocol'' for data exchange. In addition to data integration opportunities, SWTs support data quality assurance. Users of ontologies written in OWL can utilise open-source reasoners such as HermiT [6] or Pellet [7], or commercial reasoners such as Stardog[7] or RDFox [8] to perform inference and consistency checking over their RDF graphs. In addition, the SHACL language[8] (that became a W3C recommendation in 2017) allows further validation of RDF and OWL datasets against a set of conditions.

Engineering organisations using SWTs are demonstrating success in data quality assurance projects. One example is Aibel's Material Master Data Ontology used to check the consistency of flange designs in their products, citing potential savings of over 100 million euros [9]. In maintenance, SWTs have been used for data quality checking of maintenance work orders (or repair reports). In 2018, Abramovici et al. used SWTs to improve the quality of maintenance repair reports. To do this, they recognise entities in the repair reports, check for missing information in the pipeline, and check the data against a structural product breakdown [10]. In other recent work, natural language processing and an ontology

[3]Content from iFixit website (https://www.ifixit.com/) used in this paper is available under a creative commons license: (https://creativecommons.org/licenses/by-nc-sa/3.0/)

[4]MyFixit Dataset License: https://github.com/rub-ksv/MyFixit-Dataset?tab=License-1-ov-file#readme

[5]https://openai.com/chatgpt/

[6]https://www.w3.org/2001/sw/wiki/Main_Page

[7]https://www.stardog.com/

[8]https://www.w3.org/TR/shacl/

for maintenance activities was used to perform checks on the activities recorded in maintenance work orders as this check is crucial for reliability calculations [11].

C. MAINTENANCE TECHNICIANS AND DATA QUALITY

For a maintenance technician, procedures are an integral part of day-to-day work. Maintenance of industrial equipment involves technical, manual work and requires formal qualifications and years of training. Since maintenance work is hazardous, maintenance technicians should think carefully about the potential implications of each step in a procedure. Since maintenance technicians are the actualisation of procedure documentation, it is important to consider the factors that motivate maintenance technicians to think carefully and critically about the procedures that they are following.

Previous studies demonstrate that an organisation's maintenance procedure management process influences procedure compliance. Factors affecting compliance behaviour include perceived usefulness of the procedure [12], a technician's involvement in the design and implementation of procedures, as well as when procedures are perceived as logical, to the point, at the right technical level and user friendly [13]. Here is an example of the frustrations often experienced by maintainers captured in interviews by Kanse et al. [13], *"If a procedure is incorrect the expectation is that they will red pen the procedure and hand it back to the document writer and have it amended. That amendment part is where it gets stuck. The maintainer can sit there and red pen it and hand the document in, then all he sees is the next time the procedure comes out that nothing has changed so he says "why do I bother?" And that's when the maintainers switch off"*. This quote is an example of the motivational impacts on maintenance technicians if their requests for procedure corrections are ignored.

The reality of these experiences are confirmed in our recent work [3], [14] in which we examine the perceived impacts digital maintenance procedures on the work of maintenance technicians. We propose that digital tools should allow technicians to provide feedback about procedures as they are executing them. This feedback then needs to be reviewed and actioned in an agreed time frame with feedback to the maintenance technician. These quality control steps are important to technicians in supporting their sense of autonomy and task identity, thus contributing to the motivational aspects of their work.

III. THE OMPD MAINTENANCE PROCEDURE DOCUMENTATION ONTOLOGY

The OMPD Maintenance Procedure Documentation Ontology [15] is an OWL ontology that uses the ISO CD/TR 15926 Part 14 [16][9] as an upper ontology. OMPD has three design goals. First, OMPD is designed to be generic, with a minimal set of axioms to support a wide range of industrial organisations. Second, OMPD models information that organisations typically have in their current procedure documentation. Finally, OMPD is designed to answer a series of competency questions for different professional roles that typically use procedure data in their work (i.e. engineers, schedulers and technicians).

In this paper, we use the Static Procedure Ontology (SPO) module of OMPD as a target ontology. The 23 core classes in SPO are shown in Figure 1. This module includes classes and axioms for maintenance procedure documents, procedure processes, tasks, resources and hazards. Since its initial publication, SPO has been extended to also capture task description formations (such as `Image`, `Text` and `3D Models`) and the implicit information found in procedure texts (such as `MaintenanceAction` and `ItemInMaintenanceProcess` entities). Full documentation for the ontology is available on the Ontocommons Industrial Ontology Repository.[10] In line with standard ontology practice, each module is identified by use of a prefix. We use the prefixes `ompd:` to refer to SPO, `iso:` to refer to the ISO CD/TR 15926 Part 14 upper ontology (which SPO imports) and `proc:` to refer to specific application-level instances developed in this paper.

IV. OVERVIEW OF THE QUOKKA WORKFLOW

The Quokka workflow (pictured in Figure 2) consists of three modules. These are the *completion*, *consistency* and the *correctness* modules. Each module can be used in isolation or all three to realise an end-to-end semi-automated quality assurance pipeline for semi-structured data. This choice will depend on the current data maturity and procedure authoring practices in an organisation.

The first module is the *completion module*, designed for "completing" the structured representation of the data. When we say "completing", it means that we take implicit knowledge (i.e. knowledge hidden in unstructured texts) and make it explicit (i.e. thus the structured knowledge base more complete). Specifically, we perform information extraction (IE) to map entities and relationships from unstructured text in the documentation to a set of structured CSV files (suitable for serializing as OMPD-aligned OWL triples). The language model(s) to be used, the entities and relations be extracted, and the structure of the CSV output files are all determined by a configuration file (described in Section V). This means that users can choose how the completion module behaves, thus future proofing the implementation against a fast-changing natural language landscape. The completion module outputs a set of CSV documents containing structured data, extracted entities, and unique Internationalised Resource Identifiers (IRIs) applied to each datapoint. This output becomes the input to to the *consistency module*, where the data is used to generate an OWL serialization of an ontology.

[9]ISO CD/TR 15926 Part 14 has been renamed to the Industrial Data Ontology (IDO), this upper ontology has commenced the process of ISO Standardisation as of 2023 and is available online at: https://rds.posccaesar.org/ontology/lis14/ont/core/

[10]Public documentation of SPO module of OMPD: https://industryportal.enit.fr/ontologies/OMPD-SPO/

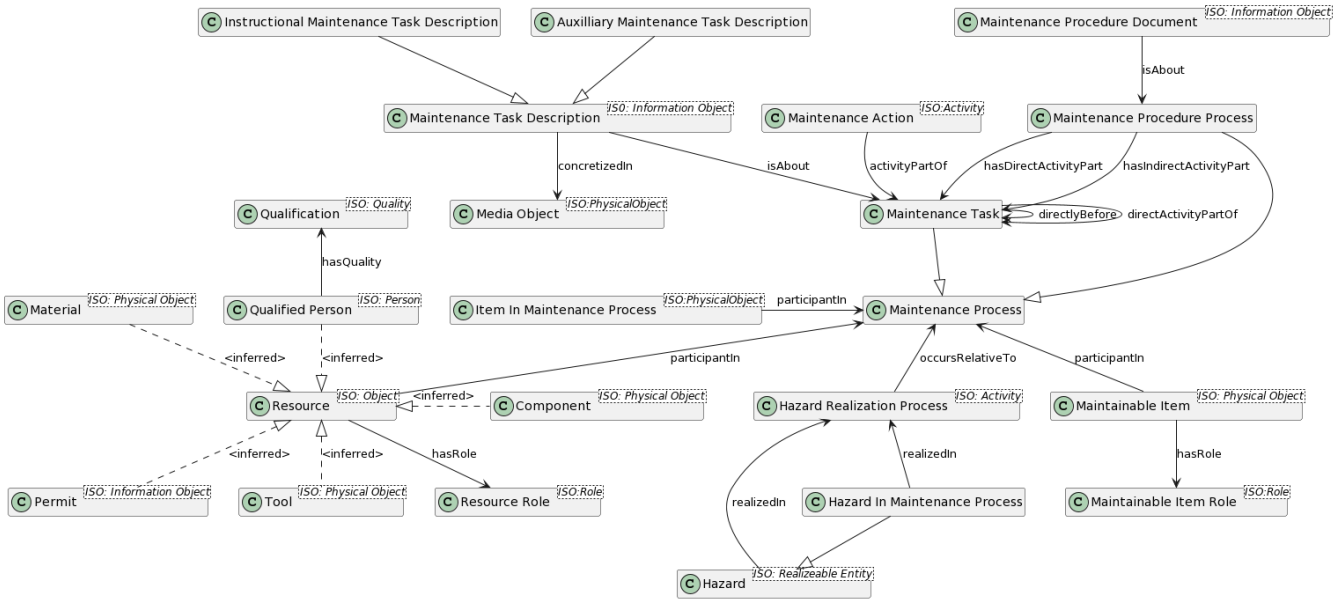

FIGURE 1. The static procedure ontology (SPO) module of the OMPD data model [15].

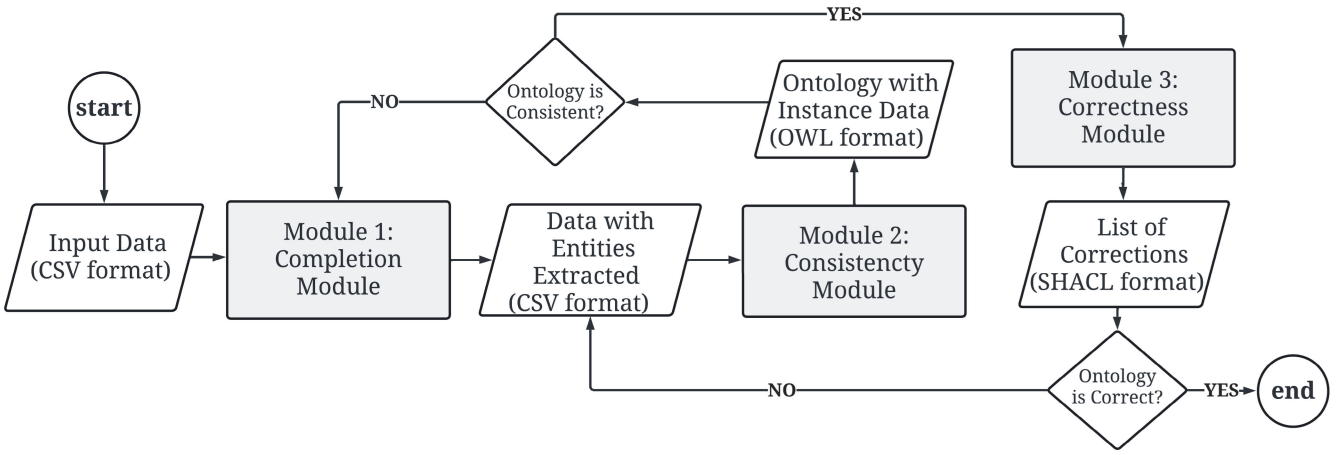

FIGURE 2. Flow diagram of the quokka workflow.

The *consistency module* uses Reasonable Ontology Templates (OTTR) [17] to convert the structured CSV files from the *completion module* into an OWL ontology, conforming with OMPD. The module uses OTTR's Lutra tool[11] to expand instance data from a CSV format into its OWL serialization. After this extraction is performed, a reasoner can be used to check the consistency of the generated ontology against the axioms of the ontology that it is aligned to. For example, using the OMPD data model (described in Section III) a Pellet reasoner (described in Section V) can check the ontology against the axioms of OMPD. An example consistency check performed in OMPD ensures that the steps in a procedure document align correctly in a sequence to ensure that there are no branching task sequences.

The final module in the Quokka workflow, the *correctness module*, takes the OWL ontology from the *consistency module* as input. This module contains two steps. First, existing RDF ontologies are imported into the ontology. These can be existing open-source models or proprietary models owned by an organisation. Second, a set of pre-configured SHACL [18] shapes are used to check the correctness of the OWL ontology (both as a standalone model and against the existing ontologies that are imported in the first step). This module is separated from the *consistency* module so that users can control their corrections to the data without generating reasoning errors. This control is important where language ambiguity in technical texts could cause

[11]The Lutra tool is provided as part of the OTTR implementation and is available at: https://gitlab.com/ottr/lutra

natural language models to produce incorrect predictions for the entity and relation annotations. Based on the outcomes of validating the ontology against a set of SHACL shapes, the *correctness module* will suggest potential data improvements to the user. Users can then update their data and re-run the *consistency module* to produce a corrected ontology.

In designing this workflow, we consider two non-functional requirements (NFR). The Quokka workflow is configurable (NFR 1) and flexible (NFR 2) to meet the needs of various organisations. These are:

- **NFR 1 - Configurability**: Users should be able to swap out the pre-configured natural language models, data models and rule sets based on their technical capability and strategic goals.
- **NFR 2 - Flexibility**: Users should be able to choose which modules of the Quokka workflow that they would like to use based on their data maturity and current procedure authoring practices.

These NFRs allow the Quokka workflow to be re-configured to use an entirely different natural language model, OTTR templates, RDF/OWL model and SHACL shapes. We implemented these NFRs to adapt the workflow to suit different industrial organisations and keep up with a fast-advancing natural language landscape into the future. The larger significance of these design choices is that our implementation of the Quokka workflow is a resource contribution for the wider semantic web community beyond maintenance procedures or technical texts. Anyone wishing to apply an information extraction model to their text-based data, align the extracted data to an ontology or validate their RDF/OWL model against existing ontologies with a set of SHACL rules can use our Python-based implementation of this workflow to accelerate their work.

V. IMPLEMENTATION

In this section, we describe how we implement each of the modules of the Quokka workflow to support semantic quality assurance of industrial maintenance procedures. Our implementation is available at: https://github.com/equonto/quokka.

A. MODULE 1: COMPLETION MODULE

Technical Language Processing (TLP) is a rapidly evolving space in the light of new Large Language Models (LLMs) and there is a growing body of research in extracting technical language, in particular from maintenance work orders [19]. In addition, [1] uses a technical language processing approach to extract entities from iFixit data in their work. Given recent developments in LLMs, we demonstrate how to leverage newly emerging models to extract implicit information from procedure documentation, in a tailored manner that meets users' requirements and strategic goals.

The *completion module* performs the process of ''completing'' the input dataset by making implicit information explicit. In industrial maintenance procedures, useful knowledge about the work performed is often locked away in unstructured texts. For example, an oil replacement procedure may contain the step: ''Use a 17 mm box end wrench to loosen the drain plug 3/4 of a turn''.[12] The text in this step tells us the activity performed (loosen), the item being worked on (drain plug) and a tool that will be required for executing a procedure (17mm box end wrench). This information is useful for planning work (i.e. what tools need to be available) and developing maintenance strategies (i.e. what types of work has been performed on this component in the past). Note that some implicit information, such as ''3/4 turn'' is not captured within the scope of this work. Future work can extend our annotation schema to capture further implicit information from the step texts depending on organizational requirements.

The input to the *completion module* is a set of CSVs describing the contents of a set of maintenance procedure documents. The input CSVs can be in any structure as long as this structure is reflected in the *completion module* configuration file. For evaluating the Quokka workflow, we use a set of four input CSVs that capture information in six iFixit[13] procedures. iFixit is a crowd-sourced repository of repair manuals [20]. Specifically, we make use of the JSON representation of preprocessed iFixit data that is publicly available [1]. We are not able to use industrial maintenance procedures as a dataset for this publication for two reasons. First, we are not able to publish the data or the produced ontology due to commercial in confidence constraints. Second, industrial organisations remain wary of using externally hosted LLMs on their own data for data privacy reasons. iFixit procedures are emblematic of procedures that are found in industry as they contain a similar style of instruction involving technical language. An example iFixit procedure that we use in this work is given in Figure 3. This procedure describes a stator replacement for a Honda CSC29 vehicle. The procedure contains a brief introduction, a structured set of tools that are required for the procedure and a set of steps (with text-based and image-based descriptions).

Maintenance procedures in industry are often stored as PDFs. While automated extraction of procedure data (from PDF into CSV format) is not the focus of this work, there is a large active body of research in this area [21], [22], [23]. Instead, the Quokka workflow uses CSVs as an input so that organisations who currently store procedure documentation in JSON, CSVs or in relational databases are supported. Users with input data in PDF or other text-based formats first need to extract their data into CSV files in a structure that is suitable for their dataset (and reflect that structure in the *completion module* configuration file). A summary of the input CSVs that we use in this work is given in Table 1.

The algorithm performed by the *completion module* is shown in Algorithm 1. The *completion module* iterates through each configuration file and transforms the input CSV specified in the configuration file into an output

[12]This procedure step text is taken from the iFixit website at https://www.ifixit.com/Guide/John+Deere+870+Oil+Change/6471

[13]iFixit: https://www.ifixit.com/

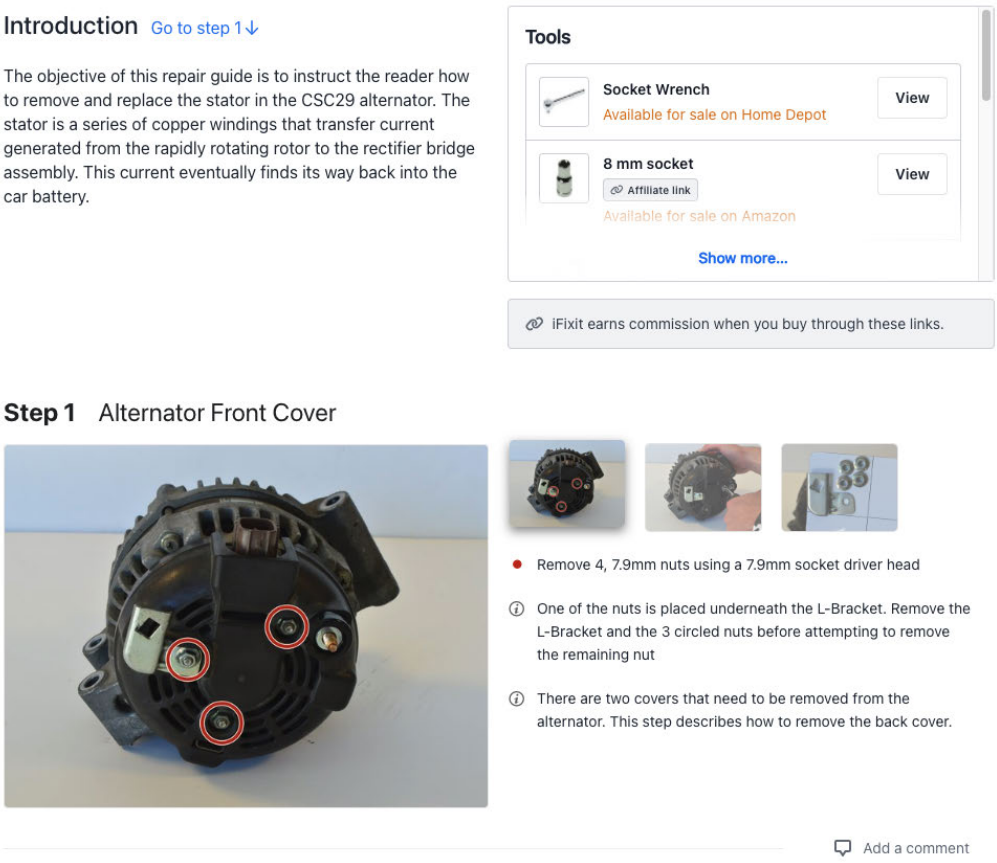

FIGURE 3. Screenshot from Honda CSC29 Stator Replacement Procedure (sourced from iFixit website at https://www.ifixit.com/Guide/Honda+CSC29+Stator+Replacement/62800).

TABLE 1. Format of input CSVs used in the *completion module*.

File name	Columns	Description	# Rows
documents.csv	procedure id, procedure name, source	Represents a list of procedure documents.	6
tasks.csv	step id, procedure id, raw text, parent process id, after task id	Represents the tasks in each procedure.	62
task descriptions.csv	task id, is instructional, type, text value, media location, is suitable for information extraction	Represents the text-based or image-based descriptions used to document the task.	271
tools.csv	tool id, process id, tool name	The structured tools found in the procedure (i.e. tools found at beginning of procedure or explicitly laid out for each step)	76

schema. If there is a text field in the input file, information can be extracted from that text field. Example extracted information is shown in Figure 4 (where a number (`n`) has been assigned to each information extraction step). First (`1`), raw entities are extracted using a *Named Entity Recognition* Model. Second (`2`), relations are determined between the entities found in the text using some *Relation Extraction* model. For procedures, information extraction is performed on text-based task descriptions. Valuable information in these task description include `activities` that are performed, the `items` they are performed on and the `tools` required to do the work. The relations that we consider in this work are `hasPatient` and `hasAgent`. This is informed by the MaintIE schema for maintenance texts [24]. These are both sub-properties of `iso:hasParticipant` in OMPD. The `hasPatient` relation is used to determine what items are being worked on in what activity (i.e. lift `hasPatient` hood). The `hasAgent` relation is used to determine what tools or materials are used to perform what activity (i.e. twist `hasAgent` wrench).

The extracted information, as well as the structured data from the input CSVs, is mapped to an output schema (represented also as a set of CSVs). In addition, entities that will form classes or instances in the ontology (produced in the next module), are given a prefix (that are used to form an IRI of the form `prefix + entity ID`). The expected structure of the input CSV and output CSVs, and the information extraction models used by the *completion*

Algorithm 1 Algorithm Performed by *completion module*

```
Data: Input CSV set
Result: Output CSV set
prepare output_folder ;
for input CSV in input CSV set do
    find configuration file ;
    if configuration file contains a information_extraction definition then
        for row in input CSV do
            prepossess text_field ;
            get entities with ner_model (if configured) ;
            relate entities with relations_model (if configured) ;
        end for
    end if
    for output_schema object in configuration file do
        serialize output ;
        write new CSV file to output_folder ;
    end for
end for
```

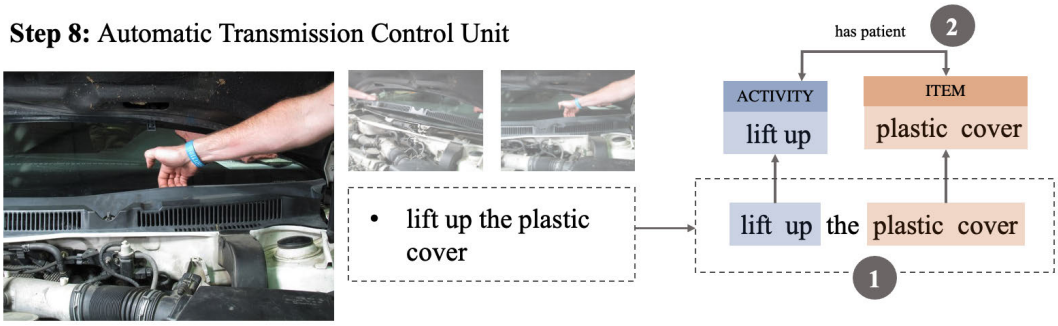

FIGURE 4. Example of information extraction performed in the *completion module* (images and text adapted from iFixit at https://www.ifixit.com/Guide/1999+2000+2001+2002+2003+2004+Volkswagen+Golf+Automatic+Transmission+Control+Unit+Replacement/2965).

module is determined according to a set of configuration files.

Example output CSVs containing one task description from Figure 4 is shown in Figure 5 (where a number (n) is assigned to the key features of the figure). Figure 5 shows one maintenance task (`proc:Task_14479`) where one maintenance task can have many maintenance task descriptions (denoted by the `0..*` multiplicity). For example, the maintenance task can be described in both a text-based description and in an image. This task has one description with the text "lift up the plastic cover" that has been assigned a unique IRI (`proc:Task_14479_description`) (1) and mapped to the `ompd:MaintenanceTaskDescription` class in the OMPD ontology. As well as assigning IRIs to the input data, implicit information has also been made explicit for this maintenance task. A `ompd:MaintenanceTask` can involve many `ompd:MaintenanceAction` entities have many `ompd:ItemInMaintenanceProcess` participants. In this case, the task involves a "lift" (2) that is performed on a "plastic cover" (3). This relationship between the "lift" action and the "plastic cover" is given in via a `ompd:hasPatient` relation (4) given in the "Action - Item Relaction" table. These output CSVs contain the explicit information that we need for mapping our data to RDF triples. In addition to this extracted information, structured information about the position of the task in the procedure is provided (5). The output of the *completion module* is the input to the *consistency module* described in Section V-B.

1) INFORMATION EXTRACTION APPROACH

In November 2022, OpenAI released ChatGPT for public use. This development brought LLMs to the forefront of both research and industrial interest in 2023 and into 2024 (the time of writing this paper). Our implementation of the Quokka workflow supports information extraction using any currently available Open AI Chat Completion Model.[14]

[14]Available models found at https://platform.openai.com/docs/models (Retrieved April 2024)

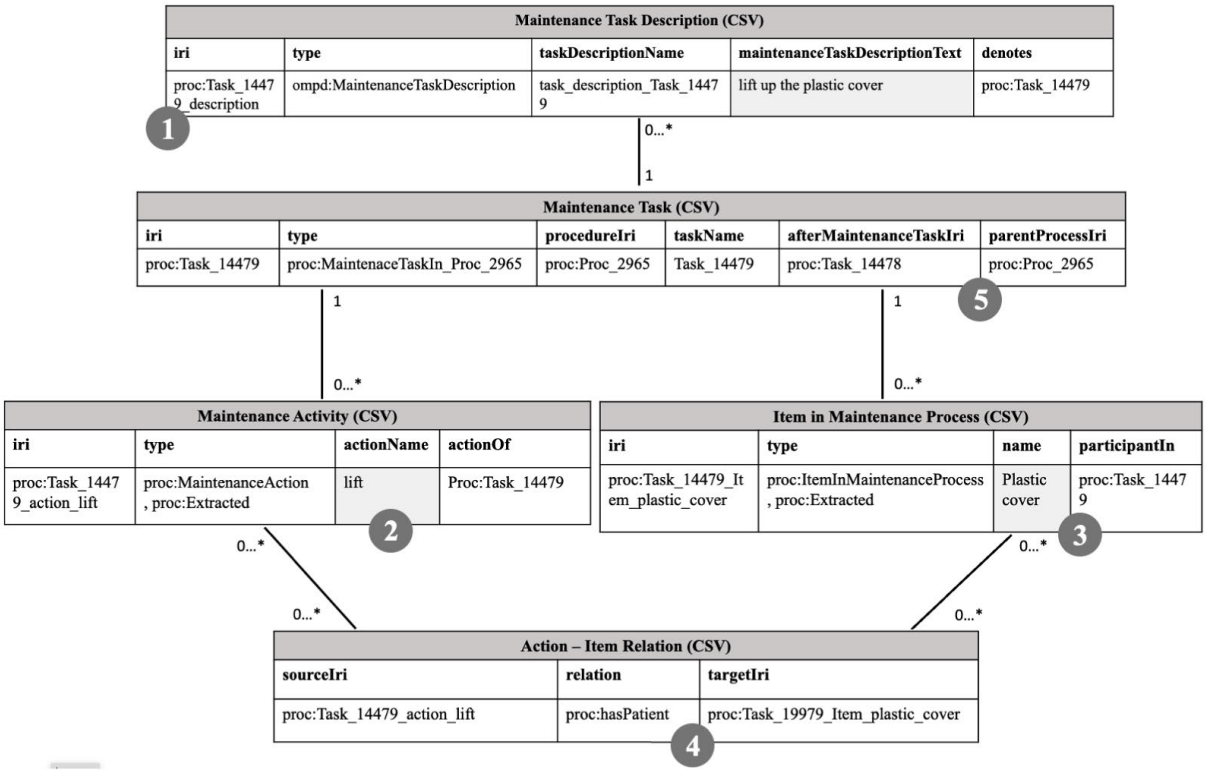

FIGURE 5. Example output data generated by the *completion module* for a single step in iFixit procedure.

OpenAI's Chat Completion models use GPT (or Generative Pre-trained Transformers) to generate text given a user's prompt. Task training on OpenAI GPT models can be performed using fine-tuning or few-shot learning. Few-shot learning is a generalised learning approach that uses only a few (often less than 10) input-output examples to train a model of its intended task. In this work, we use a few-shot learning approach. This gives organisations a mechanism to create suitable models for their data, without the need for large, curated training datasets that may not be available to the organisation. The prompts that we use for named entity recognition and relation extraction are further described in Section VI.

B. MODULE 2: CONSISTENCY MODULE

The *consistency module* maps the output CSVs from the *completion module* (Section V-A) to an OMPD-conforming OWL ontology. Once mapped to an ontology, an automated OWL reasoner checks the consistency of the data according to OMPD's axioms. For this work, we use the Pellet reasoner [7]. The Pellet reasoner is an OWL-DL reasoner based on tableaux algorithms and is available in popular ontology authoring tools such as Protégé.

Our goal in this module is to make this OWL mapping practical, configurable and repeatable (see non-functional requirements given in Section IV). For this, we use Reasonable Ontology Template (OTTR) [17]. OTTR is a language for generating OWL ontologies (and populating them with instance data) using declarative programming principles. OTTR puts data owners in control of their ontology's data, without constant involvement from a ontology expert who is proficient in Protégé or other ontology design tools. Instead, an ontology expert can setup OTTR templates, and data owners can manipulate the input data without needing to understand the details of the underlying ontology. In addition, OTTR supports ontology design pattern re-use by exposing a public template library for practitioners to re-use [17]. For these reasons, OTTR improves the efficiency and repeatability of ontology development, without the need for ad-hoc ontology population scripts that are tailored for different applications. A demonstration of the use of OTTR templates in industrial applications has been given by Lupp et al. [25].

The OTTR language provides several template formats for different expansion tasks (i.e. bOTTR,[15] stOTTR,[16] wOTTR[17]). To expand our CSV format to OWL triples we use the bOTTR and stOTTR template formats. OTTR's reference implementation, titled Lutra,[18] interprets these

[15]bOTTR (to transform tabular data to instance data in the stOTTR or wOTTR syntax): https://spec.ottr.xyz/bOTTR/0.1.2/

[16]stOTTR (syntax for defining OTTR templates - based on Turtle syntax): https://spec.ottr.xyz/stOTTR/0.1.4/

[17]wOTTR (syntax for defining OTTR templates - based on OWL syntax): https://spec.ottr.xyz/wOTTR/0.4.5/

[18]Lutra implementation: https://gitlab.com/ottr/lutra

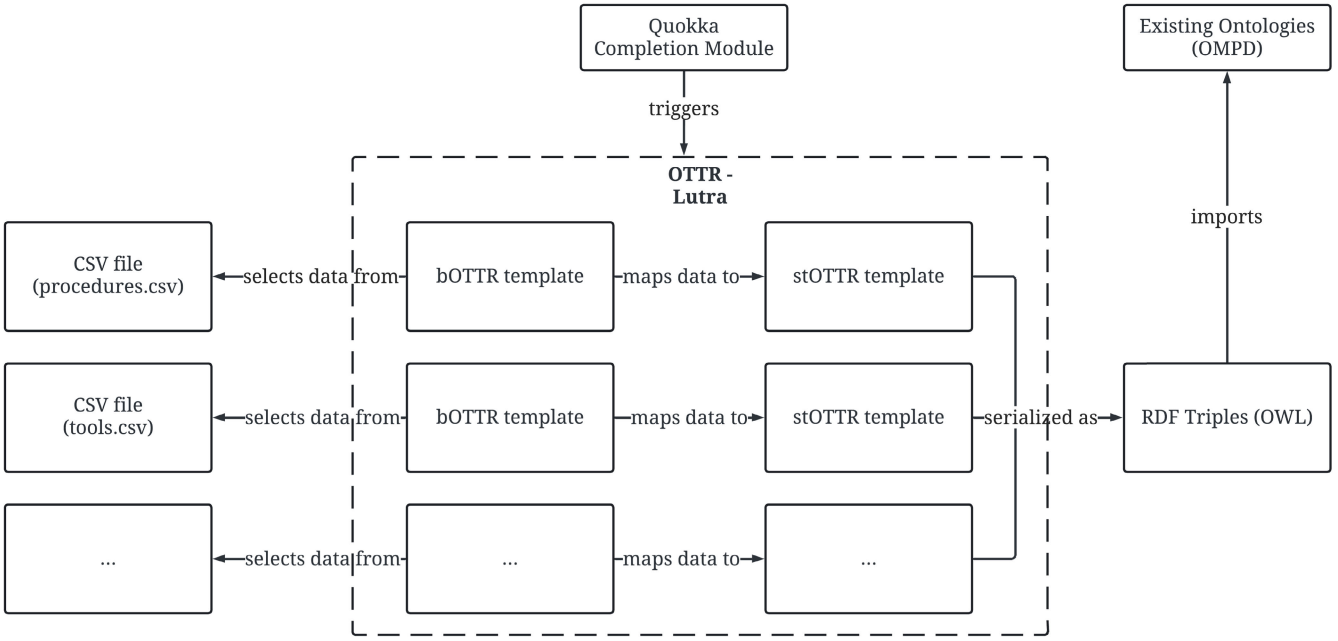

FIGURE 6. Illustration of the flow of data in the completion module (via Lutra).

templates and performs the data expansion (from CSV to OWL). The logical flow of Quokka's use of Lutra is shown in Figure 6. A set of bOTTR templates provide a mapping between the CSV files generated in the *completion module* to OTTR instances. The stOTTR templates then provide the mapping from the OTTR language to a set of OWL triples. Users of this implementation can replace the templates with their own templates without needing to change the source code. An example of the bOTTR and stOTTR templates that we use in this work are shown in Figure 7. These templates expand rows from the `maintenance_action_extracted.csv` table (i.e. the extracted `ACTIVITY` entities from the the task descriptions) and generates OWL individuals of `rdf:type ompd:MaintenanceAction`. The templates also relate the `ompd:MaintenanceActivity` with `ompd:MainteanceTask` individuals with the `iso:activityPartOf` object property. The templates that we pre-configured for this work are publicly available at https://github.com/equonto/quokka/tree/main/config/consistency/templates. In addition to the CSV data that is expanded to OWL triples, the *consistency module* imports the OMPD ontology for consistency checking against OMPD.

Once RDF triples are generated, and OMPD is imported, we use the Pellet reasoner implemented in OwlReady2 [26] to perform an automatic consistency check on the data. For example, in the OMPD ontology, the axiom $\mathit{MaintenanceTask} \sqsubseteq \mathit{MaintenanceProcess} \wedge (\leq 1\ \mathit{directlyBefore}.\mathit{MaintenanceTask})$ ensures that maintenance tasks occur in a linear sequence. Figure 8 shows both a task sequence consistent with OMPD, and a task sequence inconsistent with OMPD. Other consistency issues considered in OMPD include tasks that appear in

```
[] a ottr:InstanceMap ;
    ottr:source [ a ottr:H2Source ] ;
    ottr:query """
    select iri, actionName, actionOf, type
    from CSVREAD('staging/maintenance_action_extracted.csv')
    """ ;
ottr:template ompd-tpl:MaintenanceAction;
ottr:argumentMaps (
      [ ottr:type ottr:IRI ]
      [ ottr:type xsd:string ]
      [ ottr:type ottr:IRI ]
      [ ottr:type (rdf:List owl:Class) ]
) .
```

(a) bOTTR template for extracting maintenance actions from a CSV file

```
ompd-tpl:MaintenanceAction[
    ottr:IRI ?iri,
    ? xsd:string ?activity_name,
    ? ottr:IRI ?activity_of,
    ? List<owl:Class> ?type
] :: {
    cross | o-rdf:Type(?iri, ++?type),
    ottr:Triple(?iri, rdfs:label, ?activity_name),
    ottr:Triple(?iri, lis:activityPartOf, ?activity_of)
} .
```

(b) stOTTR template to create an OWL instrance of rdf:type Maintenance Action that is an activity part of a Maintenance Task

FIGURE 7. OTTR templates used to transform maintenance actions from a CSV into OWL instances.

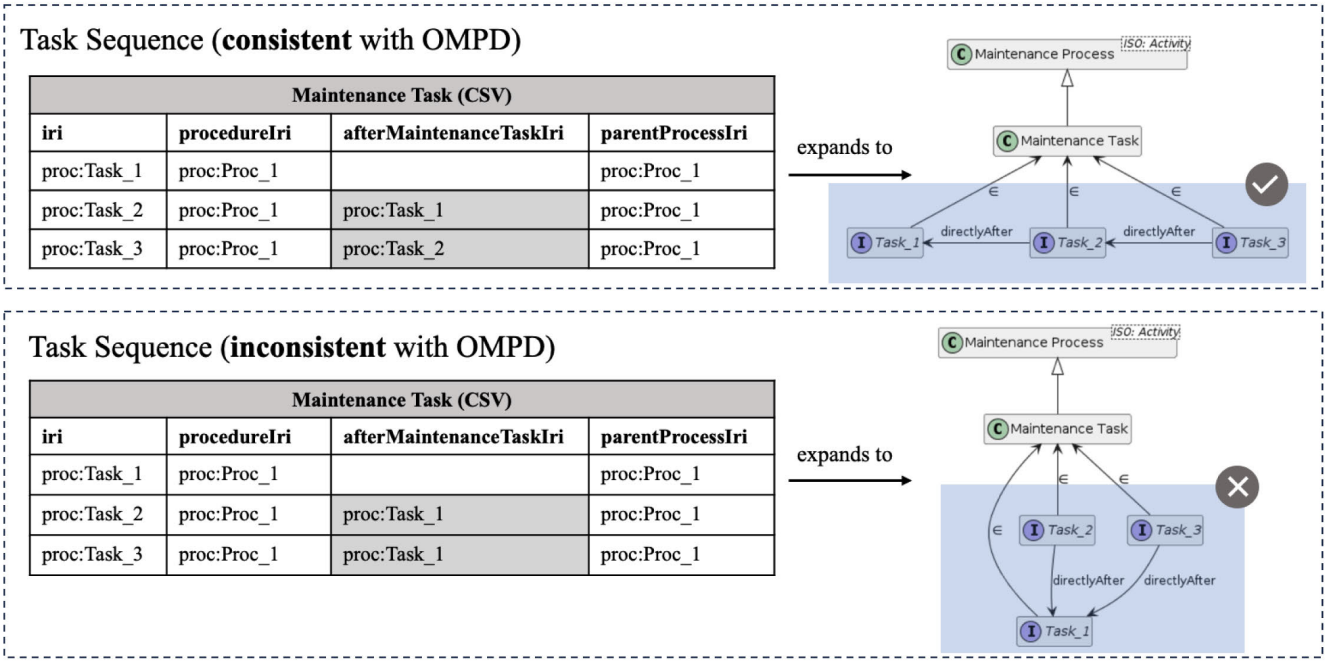

FIGURE 8. Example consistency check for task sequences using the OMPD ontology.

TABLE 2. SHACL-SPARQL shapes pre-configured in the *correctness module* for validating maintenance procedure documentation.

Shape #	Shape Name	Description	Justification
1	No Tasks in Procedure	Checks that all `ompd:MaintenanceProcedureProcess` individuals have at least one `ompd:MaintenanceTask`	All procedures should have tasks to follow and should contain at least one task.
2	Orphaned Task	Checks that all `ompd:MaintenanceTask` individuals are `activityPartOf` at least one parent (either a `ompd:MaintenanceProcedureProcess` or a `ompd:MaintenanceTask`)	All maintenance tasks should be a part of at least one procedure.
3	Task Without Sequence	Checks that if a procedure has more than one `ompd:MainteinanceTask`, that those tasks either come `directlyBefore` or `directlyAfter` another task	Ensures that the task sequence in the procedure documentation is correct.
4	Not a Maintenance Task	Checks that all `ompd:MaintenanceTask` individuals (or their children) describe at least one `ompd:MaintenanceAction`	Ensuring that all maintenance tasks correctly documented at the atomic level gives us a picture of the work being performed.
5	Action with No Item	Checks that all `ompd:MaintenanceAction` instances `hasPatient` at least one `ompd:ItemInMaintenanceProcess`	If we understand the items that actions are performed on, we can query for this
6	Tool with no Action	Checks that all `ompd:Tool` instances are `agentOf` at least one `ompd:MaintenanceAction`	If we understand the tool requirements for specific actions, we can query for this.
7	Unknown Activity	Checks that the activities identified in the text exist in an accepted list of activities. For an accepted list of activities, we use and extend the terms modelled in the existing Maintenance Activity Ontology [11].	Standardising the maintenance activities modelled from procedure texts enables us to further understand the effects of the work being performed

multiple procedures, and `ompd:Resource` entities (that are not of type `iso:Object`) incorrectly being assigned to a procedures. We provide a demonstration of typical consistency checks in Section VI.

If there are consistency issues in the ontology, users should update their input data and re-run the *completion module* to produce a consistent dataset. Using axioms in OMPD, we check the structural data quality of a set of maintenance procedures For example, we check that all `ompd:MaintenanceTask` entities are an `iso:activityPartOf` only one `ompd:MaintenanceProcedureProcess`. However, we cannot check the *content* of the procedure. For example, if the named entity recognition model used in the *completion module* (Section V-A) incorrectly tags a "engine" as an `ACTIVITY` then we cannot check for this issue. Such checks

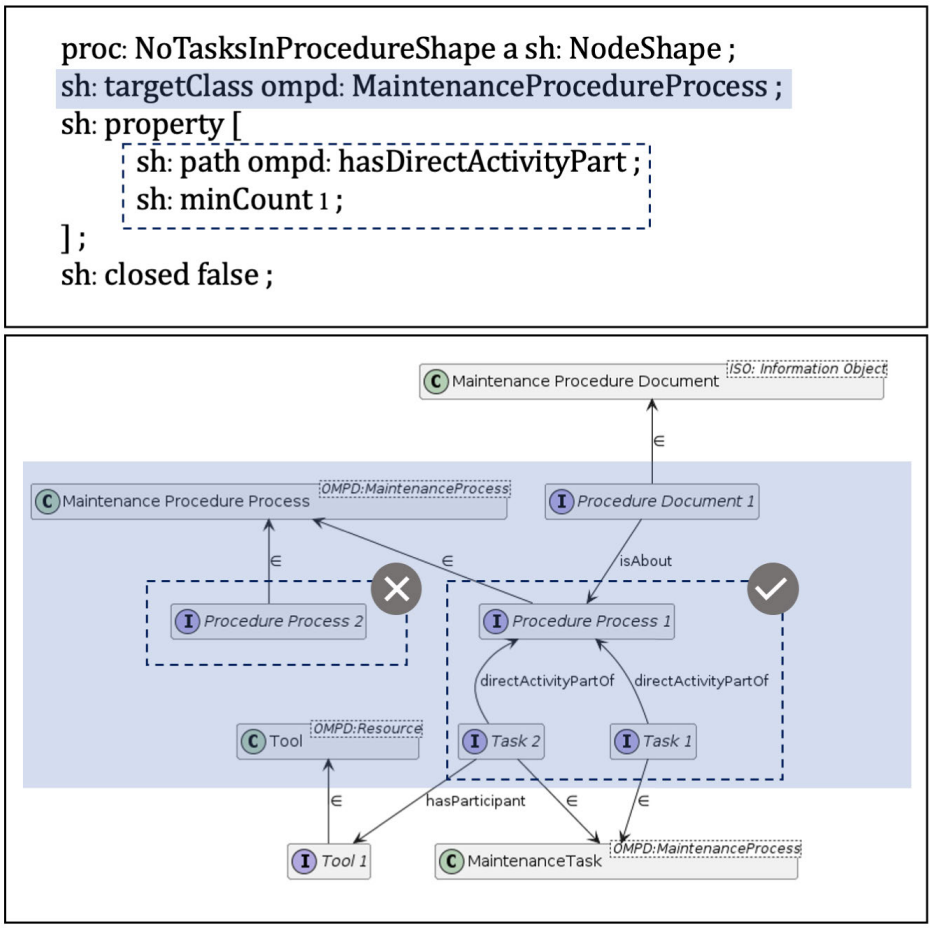

FIGURE 9. SHACL-SPARQL shape to check for instances where there is *No Tasks In Procedure*.

are, instead, performed in the *correctness module* (described in Section V-C) where closed-world constraints are applied to the data and suggestions for corrections are returned to the user.

C. MODULE 3: CORRECTNESS MODULE

In the *correctness module*, we use SHACL shapes [18] to perform closed-world constraint checking. SHACL (or the Shapes Constraint Language) is a language for validating RDF and OWL graphs that became the W3C recommendation for ontology validation in 2017. An example of a SHACL shape that we use in this work is given in Figure 9. As shown in the figure, SHACL works by first selecting entities specified in a `sh:targetClass`. In this case, the shape is selecting entities of type `ompd:MaintenanceProcedureProcess`. The shape then uses a `sh:property` definition to check the properties in the selected graph. In this case, the shape is checking if all entities have at least one `hasDirectActivityPart` relation to another entity. This is the case for `Procedure Process 1`, but not for `Procedure Process 2`. Therefore, a validation result will be returned to the user indicating there is a data quality issue in `Procedure Process 2`.

We use a SHACL extension (included in the 2017 W3C recommendation) in this work that allows the use of SPARQL in shape definitions (SHACL-SPARQL). This extension gives further expressive power to the shapes that we define. A description the SHACL-SPARQL shapes that we use to check the correctness of maintenance procedure documentation is given in Table 2. The corresponding shapes are online at https://github.com/equonto/quokka/tree/main/config/correction/shacl. Of course, it is possible to implement additional SHACL-SPARQL shapes in the *correctness module*. We have chosen to use SHACL-SPARQL so that the implementation of the Quokka workflow is configurable (NFR1, Section IV). This means that implementers of the Quokka workflow can expand this set of shapes to meet their technical requirements without modifying the source code. We have selected the presented shapes for this work to demonstrate the current capabilities of the OMPD ontology.

VI. USE AND EVALUATION

In this section, we provide a proof-of-concept demonstration of the Quokka workflow. We use this demonstration to assess OMPD's suitability for improving the semantic quality of industrial maintenance procedures. The iFixit procedures that we use in this proof of concept are listed in Table 3.

TABLE 3. Summary of Procedures Used in Evaluation.

ID	Procedure Name	# Tasks
P1	Volkswagen Golf Automatic Transmission Control Unit Replacement	11
P2	1994-2001 Acura Integra Door Panel Replacement	8
P3	2005-2007 Ford Focus Spark Plugs Replacement	9
P4	John Deere 870 Oil Change	15
P5	Disassembling 2010-2014 Ford Mustang Center Console	5
P6	Honda CSC29 Stator Replacement	14

A. SEMANTIC DATA VALIDATION AND REASONING

In this section, we demonstrate how the *consistency module* and *correctness module* perform semantic validation over procedure structure and contents. The *consistency module* maps data to RDF triples aligned with the OMPD ontology and uses a Pellet reasoner to test for consistency issues. The resulting ontology can be found at https://github.com/equonto/quokka/tree/main/data/output/ontology. The *consistency module* yields no consistency issues with our current configuration and input dataset containing six iFixit procedures. Therefore, to test for typical consistency errors that we expect to be detected in this module, we create 5 example datasets (and corresponding Quokka configurations) containing typical consistency issues that can be detected using OMPD's axioms. These datasets are described in Table 4 and can be found at https://github.com/equonto/quokka/tree/main/0_testdata. An example case is a maintenance engineer adding a branching task sequence to the dataset. This is inconsistent with OMPD that expects task sequences to be linear. In addition, the *consistency module* can check for issues with Quokka's configuration. For example, if an ontology expert (who is responsible for creating OTTR templates used in Quokka) incorrectly asserts that an instance of a tool is a *iso:Activity*.

Once the dataset mapped to RDF triples and checked for consistency, we now use the *correctness module* to perform closed-world constraint checking. In Table 5, we compare the results of the *correctness module* with an expected result determined by an SME. Shapes 1 - 3 (Table 5) that `ompd:MaintenanceTask` entities correctly sit within a procedure and have a valid sequence using closed-world constraints. For example, Shape 1 checks if there are any procedures in the dataset that contain no tasks. The *completion module* raised no correctness issues for Shape 1 - 3. These shapes correctly identify no issues in the structured information captured in the populated ontology.

Shapes 4, 5, 6 and 7 perform checks on the unstructured information in the procedure. Therefore, the results of these checks are dependant on the output of the information extraction model used in the *completion module*. The successes and limitations of this information extraction are discussed in Section VI-B. Shape 4 checks for `ompd:MaintenanceTask` entities that do not have a `ompd:MaintenanceAction` as an activity part. Quokka correctly identified one task where this is the case. In this case, the task is only described with `ompd:AuxilliaryMaintenanceTaskDescription` texts where no entities are extracted in the completion module. Therefore, there is no `ompd:MaintenanceAction` for this task in the ontology. This correctness issue can also occur if none of the maintenance task descriptions for a task have an activity term in them. When this correctness issue is raised, the task should not be considered an `ompd:MaintenanceTask` entity the maintenance engineer should consider refactoring their tasks.

Shapes 5 and 6 check for correctness issues in the procedure's unstructured text. Shape 6 checks for `ompd:MaintenanceAction` entities in the populated ontology that do not have a `ompd:hasPatient` relation to some `iso:Object` (i.e. a tool, material or item). On inspection of our human-annotated dataset, we found one row that says "look for scoring mark or cut piece of winding". Since "scoring mark" and "cut piece of winding" are tagged as `OBSERVATION` entities, there is no `ITEM`, `TOOL` or `MATERIAL` that is the patient of the "look for" activity. When the check is performed on LLM tagged dataset however, there are several cases where `ITEM` entities have been incorrectly identified and no `hasPatient` relation is tagged. In addition, the task that we identified was not identified as an error. This is because "cut piece of winding" was tagged as an `ITEM` by the LLM (where "look for" `hasPatient` "cut piece of winding"). Shape 7 is checks if there are any `TOOL` entities that do not have a `agentOf` relation to some `ACTIVITY`. On checking the human-annotated data (that we describe in Section VI-B), we found that there is one step text that reads "use a ratcheting socket wrench with an extender and a 5/8 spark plug socket to remove the existing spark plug". In the human-annotated text, the word "extender" is considered a "partOf" the ratcheting socket. In our LLM prompt, however, we do not use the partOf relation. Therefore, extender is incorrectly tagged as the agent of the remove activity (thus this issue is not identified by the SHACL shape). Instead, four cases were detected where the LLM used in the *completion module* fails to detect a `hasAgent` relation between a tool and an activity. These results are useful in checking the results of the information extraction model used in the *completion module*. However, due to issues in the information extraction model, these checks have failed to find the same correctness issues in the dataset identified by a human. For these checks to be effective, further work is required to improve the `ITEM`

TABLE 4. Checks completed for typical consistency errors detected in the *consistency module*.

Name	Description of Error	Error Type	Dataset	Detected?
Branching Task Sequence	If two `MaintenanceTask` entities appear directly after single `MaintenanceTask`, a consistency error should occur	DATA	`consistency_test_1`	✓
Invalid Reflexive Task	If a `MaintenanceTask` occurs directly after itself, a consistency error should occur	DATA	`consistency_test_2`	✓
Task Appearing in Multiple Procedures	If a `MaintenanceTask` appears in two procedures, a consistency error should occur	DATA	`consistency_test_3`	✓
Invalid Resource Role Assignment	If the Quokka configuration assigns a resource role to an entity that is not an `Object` a consistency error should occur.	CONFIG	`consistency_test_4`	✓
Invalid Hazard - Task Participation	If the Quokka configuration asserts that a entity of type `ompd:Hazard` is related to a `ompd:MaintenanceTask` via a `iso:hasParticipant` relation (where it should be realized in some `ompd:HazardRealization Process` in OMPD), a consistency error should occur.[19]	CONFIG	`consistency_test_5`	✓

TABLE 5. Suggestions made by the *correction module*.

#	Shape Name	Expected Result from Manual Check	Quokka's Result
1	No Tasks In Procedure	**No correctness issues raised** (there are no procedures with no tasks attached)	**No correctness issues raised**
2	Orphaned Task	**No correctness issues raised** (all tasks in the input dataset belong to a procedure)	**No correctness issues raised**
3	Task Without Sequence	**No correctness issues raised** (there are no broken step chains in the input sequence)	**No correctness issues raised**
4	Not a Maintenance Task	**Task_27956**	**Task_27956**
5	Action with no Item	"look for" action in "look for scoring mark or cut piece of winding")	Issues raised in 22 actions
6	Tool with no Action	"extender" tool in "use a ratcheting socket wrench with an extender and a 5 8 spark plug socket to remove the existing spark plug..."	Task_132560_object_use_wire_cutter, Task_14481_object_end_wrench, Task_7949_object_by_hand
7	Unknown Activity	**apply pressure**, **repeat this procedure**, look for, removed, off, **catch**, making sure, **spread**, pushing, **twisting**, set it aside, **pulling**, **snapping it back into place**, **turning**, **depressing**, wipe it off, **loosening**, **lifting**	**apply pressure**, **repeat this procedure**, **catch loosening**, **spread**, **lifting**, **depressing**, **turning**, **twisting**, **pulling**, **snapping it back into place**, avoid, attempting to remove, connecting, keep, reassembly, soldered back together, use, lift out, lift up, work down the side, push down, use, stick out, pushing down, begin to drain, begin, come out, wipe down, continue, drain out, add or remove, laying, engages, pop out of place, keep, even out

identification in the step texts (discussed in the following section).

Shape 7 checks for `ompd:MaintenanceAction` entities that do not match a terms list of accepted actions. For this check, we use our existing ontology for maintenance activities. Since the Maintenance Activity Ontology [11] is designed to capture the most frequently found activities in maintenance work orders, some terms are at a more atomic level. Therefore, to limit the number of false positives produced by this SHACL shape when used on iFixit data, we inspected the human-annotated texts and added terms that we consider correct but do not appear in the maintenance activity ontology to the accepted terms list. Shape 7 detected many of the issues that we found in the human-annotated text. Namely, cases where the step text is un-lemmatised, i.e. ''pulling'' should be standardised to ''pull''. In addition, it detected cases where texts are incorrectly tagged by our information extraction model. For example, we do not wish for the word ''use'' to be tagged by the information extraction model. However, this shape is able to detect cases where ''use'' has been erroneously added to the ontology. In the following section, we describe the successes and limitation of the information extraction that powers the analysis that we have described.

In summation, the *consistency module* and the *correctness module* perform data quality checks to improve the data quality of maintenance procedure documentation. These data quality improvements include structural improvements (i.e. ensuring that steps are correctly ordered and mapped to procedure documents) and content improvements (i.e. checking that maintenance tools are correctly mapped to actions that they are used to perform).

B. INFORMATION EXTRACTION

In this work, we make implicit information in the maintenance procedure texts explicit using *Named Entity*

[19] For this test, we add a new file configuration to populate entities of type `OMPD:Hazard`

Recognition and *Relation Extraction* in this work. For these experiments we use OpenAI's GPT-3.5 Turbo model. While it is not the fastest and most capable model on the market in 2024, it is a low-cost solution for experimentation. We first considered using an existing Flair model [27], however, preliminary experiments yielded poor results as the training data (maintenance work orders) document a different level of atomicity than the iFixit data. In addition, it was not possible to tag the desired TOOL, MATERIAL entities and relations with the Flair model.

For evaluation, two subject matter experts (SMEs) tagged the six procedures manually using the QuickGraph annotation tool [28]. This human-annotated dataset is publicly available and is an additional resource contribution for this work and can be accessed at https://github.com/equonto/quokka/tree/main/annotated_dataset. In the following sections, we compare the predicted results with this human-annotated data. The results for each entity type and relation type are given in Table 6. For evaluation, we give results for exact entity matching, and partial entity matching (i.e, where the entity is identified but the start and end indices of the entity do not match).

```
You are a named entity recognition assistant for
    maintenance tasks.
Respond with a comma separated list of {n-gram \\ tag}
    tuples.
The tags are:
ACTIVITY: the primary action(s) performed by a human in
    the task. It is usually one word (i.e. replace, check
    ).
ITEM: equipment as its parts (i.e. truck, switch, filter,
     connector, ignition coil, boot, hood, prop).
TOOL: a piece of tooling required for completing the task
     (i.e. paint brush, socket).
MATERIAL: a consumable item required for completing the
    task (i.e. oil, lubricant, grease, towel).
OBSERVATION: an observe state (i.e. leaking, corrosion,
    rotating, damage).
LOCATION: a position or area (i.e. right side, bottom,
    end, entrance, at the back of the engine).
CARDINALITY: a number that indicates how many of an item
    to consider (i.e. two, all, any).
SPECIFIER: adjectives describing an object or activity (i
    .e. gently, thin, new, old, remiaining, clean, dirty)
    .
NOISE: a sentence or part of a sentence that does not
    contain instructions (i.e. there are clips on the
    door).
If a token or n-gram doesn't match any tag, don't include
     a tag for it in the output list.
Use these examples: {examples here}
Q: {question}
A:
```

LISTING 1. Prompt used for entity extraction using the GPT-3.5 Turbo model.

a: NAMED ENTITY RECOGNITION

We perform named entity recognition using few-shot learning. Few-shot learning is an approach where a model learns the task from few examples, and these examples are included in a prompt supplied to an LLM. The prompt that we use includes a role (i.e. "you are a text tagging assistant for maintenance procedures") and specifies tags that the model can use (i.e. ACTIVITY, ITEM, TOOL, MATERIAL OBSERVATION, LOCATION, SPECIFIER, CARDINALITY, NOISE). The tags used in this prompt tags are labelled according to existing research in technical language processing [27]. While we only use ACTIVITY, ITEM, TOOL and MATERIAL tags within the scope of this work, we found that the model performed better when it could find an appropriate tag for entities such as "corrosion", "damage" and "left side". In earlier iterations of the prompt (where these additional tags were not used), the model would incorrectly tag these entities as ITEM or ACTIVITY. The prompt also contains 7 prompt-completion examples. We believe these prompt-completion pairs capture the intent of the given tags, while keeping the number of tokens in the prompt small. The prompt that we use (with completion pair examples omitted) is given in Listing 1 the full prompt can be found at https://github.com/equonto/quokka/blob/main/config/completion/prompt_template_ner.json.

Examples of correctly and incorrectly tagged items are given in Tables 7 and 8 respectively. ACTIVITY entities are tagged with 80% precision. However, in some cases the word "use", is tagged as an ACTIVITY (as shown in Example 1, Table 8). This is occurring despite the prompt indicating to tag it as NOISE and other texts containing the word "use" being successfully. The word "use" should not be tagged because it does not describe the work performed in the maintenance performed in the maintenance task, thus should not be added to the resulting ontology as a ompd:MaintenanceAction. The model (using the prompt provided) tagged TOOL entities with 83% precision and MATERIAL entities with 78% precision. Notice, however, when we calculate the partial entity match for these entities, both achieve greater than 90% precision. This implies that while the model can correctly identify the occurrence of a TOOL or MATERIAL it sometimes includes too many (or too little) words in the tagged n-gram. An example is given in Example 2, Table 8 where "another fixture" is tagged as a TOOL. In the human-annotated set, however, only "fixture" is tagged as a TOOL.

The model, however, struggled to tag ITEM entities sufficiently (57% precision for exact entity matches, 74% precision for partial entity matches). This is occurring for two reasons. First, in the human-annotated set, we tag items as their atomic parts where possible. This choice is consistent with existing TLP research [24]. However, the model tends to tag these as a single item. An example of such case is a "hood latch" where "hood" and "latch" should be tagged as separate items (Example 3, Table 8). Second, in the human-annotated set we do not tag parts of the sentence that are not related to the instruction (and in the prompt we tag these as NOISE). An example of such a text is "once the nut is off, you can then remove the pulley". The part of the sentence that reads "once the nut is off" is considered NOISE because it does not describe the activity occurring in the step text (i.e. "remove the pulley). However, we found that the model tended to over-tag ITEM entities that should, instead, fall into the noisy part of a sentence. These results are consistent with

TABLE 6. Comparison of six SME tagged iFixit procedures against procedures tagged by GPT-3.5 Turbo.

NAMED ENTITY RECOGNITION								
	Exact Entity Match				Partial Entity Match			
Entity	Precision	Recall	F1	Sup.	Precision	Recall	F1	Sup.
ITEM	0.57	0.64	0.60	139	0.74	0.88	0.80	139
ACTIVITY	0.80	0.88	0.83	143	0.85	0.94	0.89	143
TOOL	0.83	0.73	0.77	33	0.97	0.85	0.90	33
MATERIAL	0.78	0.78	0.78	18	0.94	0.89	0.92	18
RELATION EXTRACTION								
	Exact Entity Match				Partial Entity Match			
HAS_PATIENT	0.49	0.45	0.47	150	0.71	0.66	0.69	150
HAS_AGENT	0.48	0.58	0.53	36	0.59	0.72	0.65	36

TABLE 7. Examples of correctly tagged procedure step texts.

Step Text	Tags
Remove the rubber gasket from the rain tray	remove (ACTIVITY), rubber gasket (ITEM), rain tray (ITEM)
Use a socket wrench to release the passenger-side windshield wiper locking nut	socket wrench (TOOL), release (ACTIVITY), passenger-side windshield wiper locking nut (ITEM)
use an offset phillips screwdriver to remove a screw in the back of the automatic control unit cover	use (NOISE), offset phillips screwdriver (TOOL), remove (ACTIVITY), screw (ITEM), in the back of the automatic control unit cover (LOCATION)
loosen these two bolt with a basic socket wrench suited for 7 mm bolt	loosen (ACTIVITY), two (CARDINALITY), bolt (ITEM), basic socket wrench (TOOL), suited for 7 mm bolt (SPECIFIER)
apply an even layer of anti-seize compound to the thread of the spark plug	apply (ACTIVITY), even layer (SPECIFIER), anti-seize compound (MATERIAL), thread (ITEM), spark plug (ITEM)

TABLE 8. Examples of incorrectly tagged procedure step texts.

Example Number	Step Text	Tags
1	use compressed air to clear any debris from the area around the ignition coil on top of the engine	**use (ACTIVITY)**, compressed air (TOOL), clear (ACTIVITY), debris (ITEM), area around the ignition coil (LOCATION), on top of the engine (LOCATION)
2	use a c-clamp or another fixture to hold the pulley in place	use (NOISE), c-clamp (TOOL), **another fixture (TOOL)**, hold (ACTIVITY), pulley (ITEM), in place (LOCATION)
3	close the hood firmly and make sure that the hood latch engages	close (ACTIVITY), hood (ITEM), firmly (SPECIFIER), make sure (ACTIVITY), **hood latch (ITEM)**, engages (ACTIVITY)

existing TLP research that has struggled to tag `ITEM` entities with a high accuracy [27].

b: RELATION EXTRACTION

The prompt that we use *Relation Extraction* is given at https://github.com/equonto/quokka/blob/main/config/completion/prompt_template_relations.json. We use this prompt to find `HAS_AGENT` and `HAS_PATIENT` relations between entities and the step texts. The prompt also includes a `HAS_PARTICIPANT` relation. While we do not use this relation in the produced ontology, we found that its inclusion prevented the model from over-tagging `HAS_PATIENT`. Examples of correctly tagged texts are given in Table 9. While on visual inspection of the results, the model appears to be able use the `HAS_AGENT` and `HAS_PATIENT` relations effectively, the qualitative results that we give in Table 6 remain low. This is occurring because tagging of `ITEM` entities has a low precision. We use a strict evaluation method (where both entities and relations need to be correct for the result to be considered a true positive) thus results are detrimentally impacted by incorrectly tagged `ITEM` entities.

c: SUMMARY

These experiments signal a promising future for LLMs for information extraction for maintenance procedures and perhaps other TLP applications. However, there several opportunities for future work. An avenue for future work is to further examine the effects of prompt-engineering and fine-tuning on the use of LLMs for procedure texts. Future work includes performing experiments with alternative (including open-source) models, and assessing the impacts of using a fine-tuned model over a base LLM. There are two known limitations in our information extraction. The first limitation is that while we use a low temperature value (of 0.1) as a parameter of the models, the results of OpenAI's LLMs are inherently non-deterministic. This means that in the case of some step texts where the tagging is ambiguous (i.e. a decision to be made about whether to tag "from the rain tray" as an item or a location), the model tags the record slightly differently on each run. A possible resolution in future work is to test if fine-tuning the LLM has an impact on the deterministic nature of the results. Currently, it is possible that the base LLM is undecided between multiple interpretations of a step text and this could be improved

TABLE 9. Examples of correctly extracted relations in procedure step texts.

Step Text	Extracted Triples
pull the bottom of the door panel away from the door, and lift the entire panel off the door	pull HAS_PATIENT door panel, lift HAS_PATIENT panel
tighten the spark plug to 11 ft-lbs with a torque wrench	tighten HAS_PATIENT spark plug, tighten HAS_AGENT torque wrench
lift the engine panel up to remove it and set it aside	lift HAS_PATIENT engine panel, remove HAS_PATIENT engine panel

with a fine-tuning process. The second limitation is in our handling of plurals. The named entity recognition prompt that we use is designed such that all entities are collected once. Therefore, if the word "engine" appears multiple times in the text, then Quokka treats it as one engine when performing the ontological conversion. However, if the next says "remove all four bolts", there will still only be one "bolt" entity in the resulting tagged set (thus ontology). To improve this in future work, our prompts could be re-designed to return a cardinality of each entity in the text (i.e. bolt x4). Using this approach, the correct instances could be created in the ontology.

VII. DISCUSSION

In this section, we discuss what the impacts of the work presented in this paper on the ontology community, maintenance technicians and maintenance engineers. We also discuss several opportunities for future work.

A. WHAT ARE THE CONTRIBUTIONS FOR THE ONTOLOGY COMMUNITY?

In this work, we demonstrate the application of existing semantic quality assurance tools on a real-world industrial use case in a workflow that involves both structured information and unstructured texts. Procedures are a general concept, applying to many different domains including medicine and process engineering. The Quokka workflow is configurable (NFR1) so that other domains can use concepts from OMPD with only small configuration changes. We perform OWL reasoning and SHACL validation over both the structured information (i.e. task sequences) and unstructured information (i.e. contents of the step texts). This demonstration is made possible thanks to recent advancements in TLM and LLMs that allow for implicit knowledge to be extracted from the unstructured text found in maintenance procedures without the need for pre-annotated training data. Pre-annotated training data is time intensive and costly to create, therefore this feature is useful for practitioners who need to quickly demonstrate a proof-of-concept of their workflow. Further, the configurability of the Quokka workflow ensures that LLMs continue to be supported as the models evolve. While the information extraction presented in this paper has limitations, particularly in `ITEM` identification using few-shot learning, this demonstration is a useful starting point for organisations wishing to apply semantic technology to their industrial data.

B. HOW DOES THIS WORK HELP MAINTENANCE TECHNICIANS?

This work supports maintenance technicians in providing an extensible and repeatable quality assurance process for maintenance procedure documentation. As mentioned in Section II-B, a technician's motivational aspects of work are affected by delayed or ineffective procedure updates. In organisations where procedure documentation is paper-based and exists in non-uniform PDF templates, it is difficult to update procedures consistently. For example, if a technician points raises an issue with a documented item of tooling in a procedure document, it is difficult to find all procedures that might require the same change of tooling. While an existing procedure document can be updated, the technician is likely to see that same issue come up multiple times. This work focuses specifically on *machine-readable* maintenance documentation that allows for improved data quality checking. By validating the contents of the step texts (such as all maintenance task must have one maintenance action), we validate the form and content of the maintenance procedure. For technicians, this means that procedures are described in a uniform way and the document style is not dependant on the original author of the procedure.

C. HOW DOES THIS WORK HELP MAINTENANCE ENGINEERS?

Machine-readable maintenance procedures presents further opportunities for a maintenance engineer's work. In this work we extract `ompd:MaintenanceAction` from the step texts in maintenance procedure documentation for populating an ontology. We also present a SHACL shapes to validate these actions. For example, we define a SHACL shape to check the label of the `ompd:MaintenanceAction` against an accepted terms list. We then provide the maintenance engineer the opportunity to decide how to correct their data if validation issues are raised. This capability is significant for maintenance engineers to track *what* work was done on *what* items. This information is a useful input to maintenance strategy improvements. For example, if a filter is the patient of a "change out" activity, maintenance engineers can then trace how often that filter is changed using the frequency that the procedure is executed. If maintenance engineers can understand and track the work performed maintenance procedures, they have increased visibility over the maintenance occurring on equipment and its components.

D. WHAT OPPORTUNITIES ARISE FOR FURTHER WORK?

This work presents several opportunities for future work. First, we demonstrate examples of consistency and correctness rules that are suitable for validating maintenance procedure documentation. Future work involves discussing

the consistency and correctness rules that have highest value with domain experts.

Second, there is value in validating procedure documentation by matching the results against existing industrial vocabularies. In this work, we consider the Maintenance Activity Ontology [11] to check if the activity terms found in the procedure documentation were consistent with known terms. Upon finding that several activities in the iFixit data are documented at a more atomic level than those captured in the maintenance activity ontology, we extended the terms list from the maintenance activity ontology to also capture these terms. Future work involves analysing the effects of this extension and considering if the maintenance activity ontology can accommodate activities at a more granular level.

Finally, while we have presented a proof of concept in this work, future work involves further validation activities to test that Quokka can effectively deliver value at scale using real-world industrial data. It is important to take ethical considerations into account when using LLMs with industry data. The first consideration is confidentiality. If data is considered confidential or of proprietary value, then organisations may prefer to use open-source LLMs such as Llama[20] in secure self-contained environments. The second consideration is anonymity. If procedure documentation contains identifiable information (such as author names and email addresses) such information should be removed before passing the data to an LLM (unless prior consent is received).

VIII. CONCLUSION

In this work, we demonstrate a proof-of-concept workflow, titled *Quokka* for improving the *completeness*, *consistency* and *correctness* of machine-readable maintenance procedure documentation. For this, we use OpenAI's GPT to extract information from maintenance procedures. Our work shows success using few-shot learning to identify tools, materials and activities. However, more work is required to effectively identify items and relations using few shot learning. Next, we use OTTR to map the procedure to OWL triples (aligned with OMPD). Using OTTR, we demonstrate a configurable and repeatable way for organisations to produce and manage OWL-based procedure documentation to make the most of from consistency checking capability of OWL reasoners. Finally, we use SHACL to check the documentation against closed-world constraints.

This work has material impacts on the work of both maintenance technicians and maintenance engineers. This work can be extended to support new domains where procedures are used day-to-day. These include manufacturing, energy and surgical domains. A key contribution of this work is our publicly available code, that is designed to be both flexible and configurable to support re-use and future work in additional domains.

[20]Llama: https://llama.meta.com/

REFERENCES

[1] N. Nabizadeh, D. Kolossa, and M. Heckmann, "MyFixit: An annotated dataset, annotation tool, and baseline methods for information extraction from repair manuals," in *Proc. 12th Lang. Resour. Eval. Conf.* Paris, France: European Language Resources Association, May 2020, pp. 2120–2128.

[2] A. Kelly, *Maintenance Systems and Documentation*. Amsterdam, The Netherlands: Elsevier, 2006.

[3] C. Woods, M. A. Griffin, T. French, and M. Hodkiewicz, "Using job characteristics to inform interface design for industrial maintenance procedures," in *Proc. CHI Conf. Human Factors Comput. Syst.*, May 2021, p. 180.

[4] A. Pliatsios, K. Kotis, and C. Goumopoulos, "A systematic review on semantic interoperability in the IoE-enabled smart cities," *Internet Things*, vol. 22, Jul. 2023, Art. no. 100754.

[5] A. Rhayem, M. B. A. Mhiri, and F. Gargouri, "Semantic web technologies for the Internet of Things: Systematic literature review," *Internet Things*, vol. 11, Sep. 2020, Art. no. 100206.

[6] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "HermiT: An OWL 2 reasoner," *J. Automated Reasoning*, vol. 53, no. 3, pp. 245–269, Oct. 2014.

[7] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *J. Web Semantics*, vol. 5, no. 2, pp. 51–53, Jun. 2007.

[8] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee, "RDFox: A highly-scalable RDF store," in *Proc. Int. Semantic Web Conf.* Cham, Switzerland: Springer, 2015, pp. 3–20.

[9] M. G. Skjæveland, A. Gjerver, C. M. Hansen, J. W. Klüwer, M. R. Strand, A. Waaler, and P. Ø. Øverli, "Semantic material master data management at aibel," in *Proc. 17th Int. Semantic Web Conf. (ISWC) Posters Demonstrations, Ind. Blue Sky Ideas Tracks*, 2018, Art. no. 90.

[10] M. Abramovici, P. Gebus, J. C. Göbel, and P. Savarino, "Semantic quality assurance of heterogeneous unstructured repair reports," *Proc. CIRP*, vol. 73, pp. 265–270, Jan. 2018.

[11] C. Woods, M. Selway, T. Bikaun, M. Stumptner, and M. Hodkiewicz, "An ontology for maintenance activities and its application to data quality," *Semantic Web*, vol. 15, no. 2, pp. 319–352, Apr. 2024.

[12] X. Hu, M. Griffin, G. Yeo, L. Kanse, M. Hodkiewicz, and K. Parkes, "A new look at compliance with work procedures: An engagement perspective," *Saf. Sci.*, vol. 105, pp. 46–54, Jun. 2018.

[13] L. Kanse, K. Parkes, M. Hodkiewicz, X. Hu, and M. Griffin, "Are you sure you want me to follow this? A study of procedure management, user perceptions and compliance behaviour," *Saf. Sci.*, vol. 101, pp. 19–32, Jan. 2018.

[14] C. Woods, M. Hodkiewicz, T. French, and P. Griffin, "Assessing user interface design features using the job characteristics model in industrial maintenance," 2024.

[15] C. Woods, T. French, M. Hodkiewicz, and T. Bikaun, "An ontology for maintenance procedure documentation," *Appl. Ontol.*, vol. 18, no. 2, pp. 169–206, Aug. 2023.

[16] *ISO 15926-14: Industrial Automation Systems and Integration—Integration of Life-Cycle Data for Process Plants Including Oil and Gas Production Facilities—Part 14: Data Model Adopted for OWL 2 Direct Semantics*, ISO/CD Standard 15926-14, 2003.

[17] M. G. Skjæveland, D. P. Lupp, L. H. Karlsen, and J. W. Klüwer, "OTTR: Formal templates for pattern-based ontology engineering," in *Advances in Pattern-Based Ontology Engineering*. Amsterdam, The Netherlands: IOS Press, 2021, pp. 349–377.

[18] H. Knublauch and D. Kontokostas. (2017). *Shapes Constraint Language (SHACL)*. Accessed: May, 28, 2024. [Online]. Available: https://www.w3.org/TR/shacl/

[19] M. P. Brundage, T. Sexton, M. Hodkiewicz, A. Dima, and S. Lukens, "Technical language processing: Unlocking maintenance knowledge," *Manuf. Lett.*, vol. 27, pp. 42–46, Jan. 2021.

[20] G. Getto and J. T. Labriola, "iFixit Myself: User-generated content strategy in 'the free repair guide for everything,'" *IEEE Trans. Prof. Commun.*, vol. 59, no. 1, pp. 37–55, Jan. 2016.

[21] S. S. Paliwal, D. Vishwanath, R. Rahul, M. Sharma, and L. Vig, "TableNet: Deep learning model for end-to-end table detection and tabular data extraction from scanned document images," in *Proc. Int. Conf. Document Anal. Recognit. (ICDAR)*, Sep. 2019, pp. 128–133.

[22] S. Schreiber, S. Agne, I. Wolf, A. Dengel, and S. Ahmed, "DeepDeSRT: Deep learning for detection and structure recognition of tables in document images," in *Proc. 14th IAPR Int. Conf. Document Anal. Recognit. (ICDAR)*, vol. 1, Nov. 2017, pp. 1162–1167.
[23] M. Zhu and J. M. Cole, "PDFDataExtractor: A tool for reading scientific text and interpreting metadata from the typeset literature in the portable document format," *J. Chem. Inf. Model.*, vol. 62, no. 7, pp. 1633–1643, Apr. 2022.
[24] T. K. Bikaun, T. French, M. Stewart, W. Liu, and M. Hodkiewicz, "MaintIE: A fine-grained annotation schema and benchmark for information extraction from maintenance short texts," in *Proc. Joint Int. Conf. Comput. Linguistics, Lang. Resour. Eval. (LREC-COLING)*, 2024, pp. 10939–10951.
[25] D. P. Lupp, M. Hodkiewicz, and M. G. Skjæveland, "Template libraries for industrial asset maintenance: A methodology for scalable and maintainable ontologiesy," in *Proc. 13th Int. Workshop Scalable Semantic Web Knowl. Base Syst., 19th Int. Semantic Web Conf. (ISWC)*, vol. 2757, 2020, pp. 49–64.
[26] J.-B. Lamy, "Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies," *Artif. Intell. Med.*, vol. 80, pp. 11–28, Jul. 2017.
[27] M. Stewart, M. Hodkiewicz, W. Liu, and T. French, "MWO2KG and echidna: Constructing and exploring knowledge graphs from maintenance data," *Proc. Inst. Mech. Eng. O, J. Risk Rel.*, pp. 1–13, Nov. 2022.
[28] T. Bikaun, M. Stewart, and W. Liu, "QuickGraph: A rapid annotation tool for knowledge graph extraction from technical text," in *Proc. 60th Annu. Meeting Assoc. Comput. Linguistics Syst. Demonstrations*, 2022, pp. 270–278.

CAITLIN WOODS received the Ph.D. degree in digital transformation of industrial maintenance procedures. She is currently working in software engineering and has industry experience across the mining, manufacturing, and technology sectors. She specialized in human–computer interaction and ontology engineering, studying how both fields can come together to transform the way that organizations create, manage, and use procedure documentation. She is a member of the Industrial Ontologies Foundry Maintenance Working Group and WG26 on the ISO CD/23726 automation systems and integration-ontology-based interoperability standard.

MELINDA HODKIEWICZ (Member, IEEE) is currently a Professor with the School of Engineering, The University of Western Australia. She works in the field of applying natural language processing, knowledge graphs, and ontologies to industrial maintenance and safety data. Her background includes more than ten years in industry in production and maintenance roles followed by a career in academia focusing on technical and management issues in maintenance, reliability, and safety. In 2019, she was made a fellow of Australian Academy of Technology and Engineering. She was Australia's Nominated Representative on the ISO/TC251 55000-2 Committee for Asset Management, from 2010 to 2014. She sits on ISO/TC184/SC 4 Industrial Data responsible for ISO/CD 23726 Automation Systems and Integration-Ontology-Based Interoperability and the Technical Oversight Board of the Industrial Ontologies Foundry. In 2016, she was awarded the MESA Medal The Lifetime Achievement Award for services to the asset management community in Australia.

TIM FRENCH is currently an Associate Professor with The University of Western Australia, working in the fields of logic, artificial intelligence, and knowledge representation and reasoning. His particular areas of research interests include reasoning about uncertainty and modeling probabilistic reasoning. He has also worked extensively on awareness in multiagent systems, probabilistic reasoning in games, and reasoning about informative updates in multi-agent systems. These include axiomatizations, decision procedures, and expressivity results. He also works on applying these ideas in industrial settings. He has worked on projects using automated planning for industrial processes, capturing tacit and implicit knowledge from domain experts, and machine learning for complex processes. He is also the Regional Contest Director of the South Pacific Programming Contest and a sub-contest of the Inter-Collegiate Programming Contest.