

# Hybrid AI Approach for Knowledge Graph Construction

Frank Wawrzik<sup>1,\*</sup>, Rami Dhouib<sup>1,†</sup>, Khushnood A. Rafique<sup>1</sup> and Christoph Grimm<sup>1,\*</sup>

<sup>1</sup>RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau, Gottlieb Daimler Straße, Kaiserslautern, 67663, Germany

## Abstract

Ontology development is becoming increasingly mature and as a consequence the question and challenge arise on how to handle larger amounts of knowledge and data in automated ways. Current approaches on constructing knowledge graphs are based on machine learning, especially natural language processing. They are improving in quality and increasingly add quality assurance aspects to ensure better usability of the data and semantics for machines. However, none of those currently make use of consistency checking in order to ensure adherence to a consistent schema. In this work, we provide a new approach that allows for continuous and better quality assurance by integrating reasoner and human in the loop into the knowledge graph construction process. We show our approach, system and an evaluation of the results of our system engineering and electronics knowledge graph. The results show an improved knowledge graph construction according to the metrics of conciseness (succinctness) and consistency (coherency) of up to 67,4 % and our method decreases development time to train the neural network.

## Keywords

hybrid AI, knowledge graph construction, automated consistency checking, AI and human in the loop, natural language processing, machine learning, OWL, electronic knowledge graph

## 1. Introduction

Ontologies are a central part of semantic web and knowledge management. They provide a structured and formal representation of knowledge, which enables machines to reason about this knowledge. In the context of semantic web, ontologies serve as a common vocabulary for describing concepts, relationships, and properties of various domains. Constructing knowledge graphs is a challenge because (1) we are dealing with large amounts of data which are hard to oversee, supervise and understand. And because (2) the quality of those graphs is often not good from the ontology engineers perspective. Usually graphs generated by for example GraphGPT [1] are vast, but lack proper structures to make them coherent. Examples are

---

*KGR4XAI@IJKKG'23: The 2nd International Workshop on Knowledge Graph Reasoning for Explainable Artificial Intelligence co-located with 12th International Joint Conference on Knowledge Graphs (IJCKG 2023), December 08–09, 2023, Tokyo, Japan*

\*Corresponding author.

<sup>†</sup>These authors contributed equally.

✉ wawrzik@cs.uni-kl.de (F. Wawrzik); dhouib@rhrk.uni-kl.de (R. D. ); khushnood.rafiq@cs.uni-kl.de (K. A. Rafique); grimm@cs.uni-kl.de (C. Grimm)

🌐 <https://cps.cs.uni-kl.de/mitarbeiter/frank-wawrzik-msc> (F. Wawrzik);

<https://cps.cs.uni-kl.de/mitarbeiter/christoph-grimm-prof-dr> (C. Grimm)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

connection with proper super classes or correct taxonomies. One of the ways going forward is creating the ontology by an ontology or knowledge engineer with care, expertise, background information and feedback from applications. And then construct a knowledge graph based on that conceptualization by machine learning techniques. In this way, it is possible to acquire data fast, automated and easily. However, the machine usually, though using context by the patterns recognized by the neural model, still usually lacks context and many errors ensue and low data quality pertain. It is difficult to assess whether the machine made an error or if the content of the constructed graph is either correct or matches with the intention or the context of the processed files and data. Here, reasoning approaches based on OWL DL can provide (1) aid in understanding errors, (2) improving the quality of the domain ontology or knowledge graph and (3) correcting errors.

Reasoning in the Semantic Web usually has three purposes:

1. checking for concept consistency of definitions or in general of classes and their relationships for example with complex class expressions
2. adding implicit knowledge via axioms with for example transitive object properties, hasValue restrictions, inverse object properties and others
3. adding if-then conclusion via rules which can be qualitative, but also quantitative with for example SWRLs Math Built-In

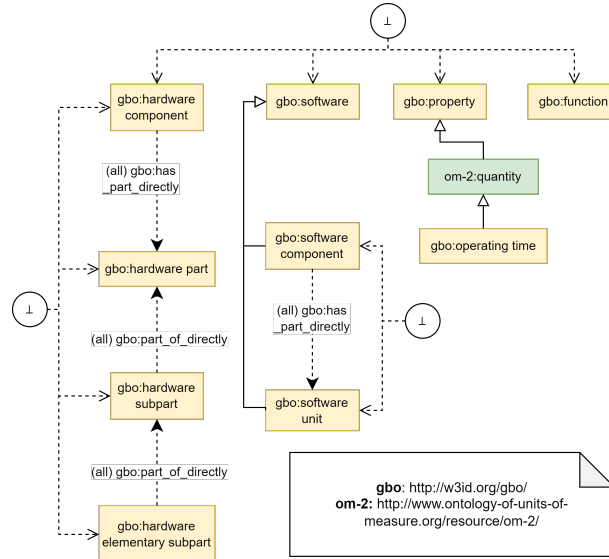
In this paper we apply reasoning in the context of purpose (1). We construct a knowledge graph based on a vocabulary called GBO (GENIAL! Basic Ontology) [2]. The construction process is further based on an NLP transformer which was trained based on the vocabulary and a wikipedia dataset [3]. Based on this prior work, we now applied the reasoning on the vocabularies definitions in a novel approach described in detail in section 4. The novelty of the approach is that it allows for continuous knowledge graph quality assurance in the loop with concise definitions. At the same time it improves the dataset and accuracy of the tagging. Which thus also improves the training of neural networks, such as transformers or large language models. The technical contribution of this paper is the integrated architecture of reasoning and machine learning, which is both enhanced with the computer and the user. The results show that such methodology provides a beneficial addition to the knowledge construction pipeline and can both improve the quality of the graph as well as reduce work for knowledge engineers.

## 2. Background

### 2.1. Knowledge Base: GENIAL! Basic Ontology (GBO)

Basic conceptualization of the microelectronics domain in regard to automotive. Basic classes include hardware, software, systems, quantities, functions, dependencies, roadmap and context knowledge. GBO is based on BFO and the ISO26262 standard in electronic safety. Figure 1 is a simplified and shortened version of the relevant classes and axioms to understand the approach in this paper. As we can see, both hardware and software are structured into hierarchical levels. Each level is the direct next hierarchical level, indicated by the "has\_parts\_directly" object property. Further, all software and hardware hierarchical levels are disjoint. Further hardware, software, properties and functions are also each disjoint classes. It needs to be noted that the

parts are structured with "only" or universal (all) restrictions, which are more restricting and make the approach in section 4 more effective. Figure 1 as well as 7 were created with Chowlk Visual Notation. For symbol clarification we refer to the documentation.<sup>1</sup>



**Figure 1:** Relevant classes and reasoning axioms for approach

## 2.2. Dataset

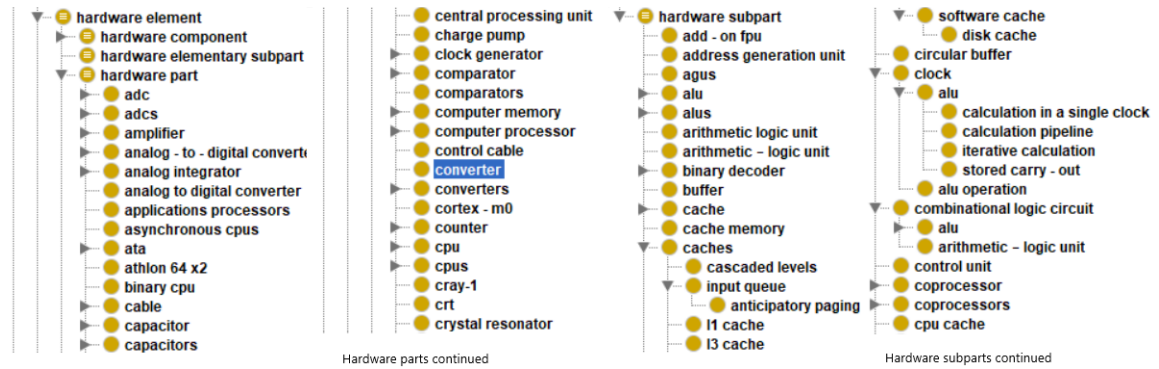
We used datasets from related articles in the domain for both tagging for Bi-LSTM neural network as well as automatic transformer-based methods. These include Wikipedia articles like "central processing unit", "analog digital converter", "automatic cruise control", and comprises currently around 20 articles. The articles were selected to represent the domain of electronics and its most prevalent terms.

## 2.3. Microelectronics Knowledge Graph

Combining the ML methods and our ontology reference model we construct our microelectronics knowledge graph according to the methodology in [3]. Figure 2 shows an excerpt of the knowledge base of hardware parts and hardware subparts. We can see various well known parts like cpus, caches and converters and memories. But also very domain related ones known only to experts, like adcs, amplifiers, clocks, alus, buffers and capacitors. The converter class is highlighted as we will have a closer look at it later in the analysis and after applying our presented method.

This paper is structured as follows: In section 3 we introduce state of the art in the related field and draw comparisons and conclusions to our work. In 4 we show the approach and some

<sup>1</sup><https://chowlk.linkeddata.es/notation.html>



**Figure 2:** Sample of classes in the hardware category

implementation details, in section 5 we show the results, outline an evaluation and finish with a discussion. In section 6 we conclude and mention future work.

### 3. State-of-the-Art

Building knowledge graphs either manually with a systematic requirements and knowledge engineering approach or automatically using algorithms and programming or machine learning is a focus of past and current research [4]. McGuinness guide in [5] or the pizza tutorial<sup>2</sup> are the beginners first go to, but often knowledge engineers have to go through many hand picked papers in order to develop their skills and expertise.

Some approaches do automate the construction process in using machine learning approaches as in [6]. They populate an ontology with a relation extraction to a certain domain from text. Another approach in [7] constructed ontologies from online ontologies. Further ontology ranking, mapping, merging and evaluation were used in addition and in complement. Automating ontology creation can increase ontology adoption, use and put creating knowledge graphs in the hands of many due to its added value. Those ontologies however may suffer from inconsistencies and contradictions. So reasoning is necessary to assure a certain coherence and quality for and of the ontology. Using OWL also provides well defined and formal semantics to represent logical base assumptions of knowledge. OWL is relatively expressive and makes reasoning on it available through well know reasoners like Konclude [8] and Pellet.

Hybrid AI approaches with neural networks and OWL-DL 2 reasoning like proposed in this work are rare. This and the next paragraphs establish a representative baseline and context for current comparable works. [9] uses machine learning to summarize text and combines this with a measure to calculate several centrality metrics, which improves finding and selecting the best node / class. Instead of using metrics, we employ OWL-DL 2 reasoning to find more accurate classes and solve contradictions. Some related work employs graph embeddings for neural networks [10, 11]. Others usually construct the knowledge graphs without reasoning. [12, 13] are some selected examples. [14] does a kind of reasoning in an interactive navigation

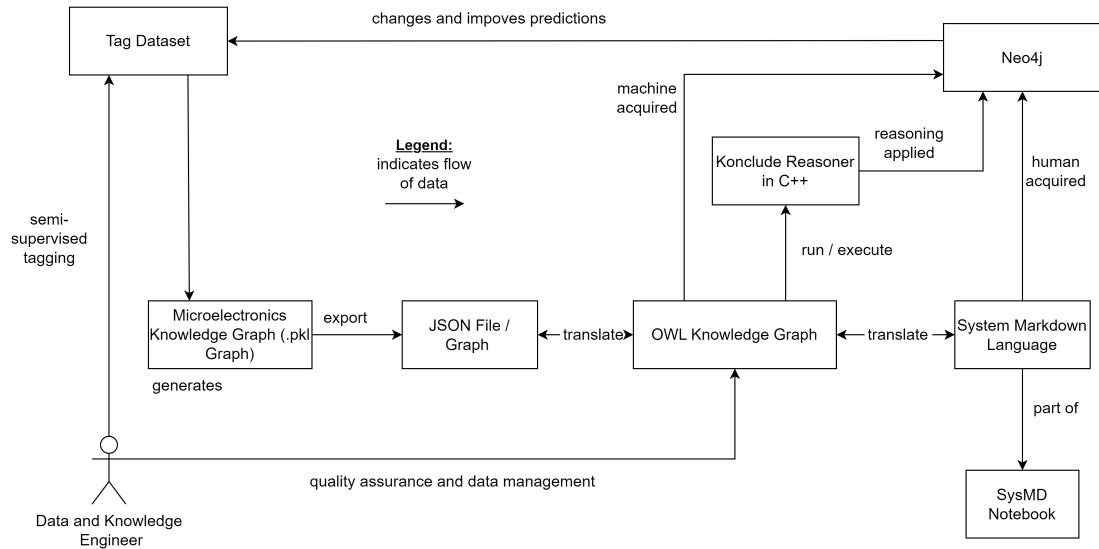
<sup>2</sup>[https://drive.google.com/file/d/1A3Y8T6nIfXQ\\_UQOpCAr\\_HFSCwpTqELeP/view](https://drive.google.com/file/d/1A3Y8T6nIfXQ_UQOpCAr_HFSCwpTqELeP/view)

robot system with question answering by matching a commonsense ontology with a home ontology and inferring relationships and classes.

There are various reasons that approaches like this one are rare. To name probably the first one is reasoning speed with large graphs and networks which is a challenge [15]. Further most ontologies are not build expressively and are rather used as domain models, where the atomic class is the central part. For example, [16] and [17] give an overview of reasoning over knowledge graphs and there are other contributions, which focus on different kinds of reasoning, e.g. rule based reasoning [18]. Entity prediction, where we find the authors combine either LSTM [19] or reinforcement learning with reasoning over graphs [20] has been developed. Further there are approaches from relation prediction [21] to query based reasoning [22]. Bringing quality assurance into large graphs is a major and challenging task and is currently emerging in recent contributions like [23]. In our work we refer to the quality metrics mentioned in<sup>3</sup>. Our approach contributes to quality assurance in the metrics of succinctness and coherency. It can be classified as hybrid and with machine learning and ontology-based reasoning with human-machine interaction. The content of language models can be directly crawled and translated [24] as well and needs to be mentioned as related work.

## 4. Approach and Method

### 4.1. The concept and overall framework



**Figure 3:** Architecture of hybrid approach with human in the loop

In this section we will go step by step through the concept and the framework. Figure 3 illustrates the approach. We start in the middle with the knowledge graph stored in OWL. The

<sup>3</sup><https://www.emse.fr/~zimmermann/KGBook/Multifile/quality-assessment/>

graph needs to be represented in OWL in order to execute reasoning on it. We translate from two formats into OWL. On the right to the knowledge graph we see the "System Markdown Language" [25]. This language is based on Markdown and a SysML v2 Metamodel and mapped to OWL. It serves system modelers as input and access to the knowledge graph with the frontend AGILA. With the tool specifications can be defined and constraint analysis calculated. Now the reasoner (Konclude [8]) is applied to the graph. All the graph contents are then represented and saved in the Neo4J database <sup>4</sup> (with help of the neosemantics library). This results in several single, but also overlapping graphs. Table 1 gives an overview and explanation. We distinguish between the complete graph, a consistent graph, which pass the reasoner tests and an inconsistent graph, which causes errors in the consistency check of the reasoner. Additionally there are unclassified classes, which are not assigned. The human acquired part (arrow on the right) is not considered here, as it is out of scope of this paper. Since the graph is partitioned now, we do know that there are errors in the graph and can correct them. The classes from the inconsistent knowledge graph are then removed from the tagging dataset in the next step. This step is essential to let the transformer / bi-LSTM improve its predictions automatically. More details on this are given in the separate section 4.4. Additionally, the user can view the inconsistent graph and make corrections directly in protege. The ontology file is connected to the dataset .csv table file and automatically updates changes by the user. The table file follows a Part-of-Speech (POS) Tagging structure, which aids the neural networks training in recognising context.

The transformer finally based on the dataset generates predictions on new input text and with a python library creates a pickle (.pkl) graph. We export the pickle graph into JSON. The second format then is JSON, which again gets translated into the OWL knowledge graph. Relationships that are generated from JSON to OWL are all established using the "some" restriction. In combination with the disjoint classes and the "only" restrictions from the vocabulary the reasoning is accomplished. For the translation we used Owlready2, which is similar to the OWL-API, but based on python instead of Java. This then closes the loop. The output of our natural

<sup>4</sup><https://neo4j.com/>

**Table 1**  
Graph Partitioning

Graph Partition	Partition Description
Basic Formal Ontology	top-level ontology to distinguish fundamental concepts like objects, processes and functions
GENIAL! Basic Ontology	electronic domain ontology to describe hardware, software and systems, properties, functions, but also dependencies and mechanical parts. Contains all axioms
complete graph	complete NLP generated graph of our current dataset, containing classes, instances and object properties
consistent graph	contains entities that pass the Konclude / Pellet consistency reasoning
inconsistent graph	contains entities that do not pass the consistency reasoning
unclassified graph	contains classes and individuals which are not allocated to a GENIAL! Basic Ontology (GBO) class

language processing (NLP) model provides us with triples comprising a head, type, and tail. Additionally, we leveraged the results of the tagging process from the GBO ontology to establish basic subclass/superclass relations. For example, classes that were tagged were assigned those tags as superclasses in the graph. Basically, if the term `cpu` is tagged as a hardware part, then the term `cpu` will have the hardware part class from GBO as its parent class. Some other subclass assignments were established that create obvious taxonomic relationships. The following sums up the approach for generating graphs:

1. Feeding the transformer model REBEL our dataset
2. Adding the classes, instances and object properties between classes to our ontology
3. Reason on the ontology to extract the machine-generated consistent ontologies
4. Store the parts of the graph in OWL files as well as Neo4j database
5. Update the dataset according to the reasoning results

## 4.2. Natural language processing

Wikipedia articles and other scientific text pieces, that dealt with the microelectronic domain, were used to train and classify text based on the GBO classes. The first step was to merge the results of the tagging system and the generated triples. The triples were first generated using a bidirectional long short-term (bi-LSTM) neural network model. Bi-LSTM models allow inputs to travel backwards and forward. The hidden layers of the network are connected to the same output. The task was divided into two parts. The first part was to classify sample occurrences into our predefined classes. This task could be referred to as Named Entity Recognition (NER) - with our custom GBO classes. The second step was to generate a graph that aligned with the relationships in the GBO. The NER task used the bi-LSTM model with 5 layers and 21 different types of labels. All hyper-parameters such as the LSTM units and dropouts were adjusted with experimentations. The training was carried out for 50 epochs with a batch size of 32. We reached a validation accuracy of 94%. With the predictions of the bi-LSTM, we could generate graphs. The graphs generator used a rule-based model. We defined a "context" that represented a small part of the text that kept the interrelated conditions intact. We settled for a context size of around 10 sentences. The identified (or the predicted) words were then extracted from the sentences. A simple rule-based model (consisting of a few if-else blocks) was then used to create relationships between these extracted words [3].

We also used the transformer to generate graphs. This gave us a better insight into the results produced by the bi-LSTM model. It also helped us compare and choose the better option in the end. The transformer [26] has surpassed other neural network models, including the convolutional neural network (CNN) and the recurrent neural network (RNN) for NLP and language generation tasks. Long-range sequence characteristics are easily captured, and the architecture scales well with both the quantity of the training data and the size of the model being trained.

In this method, we converted an original sentence comprising entities and implicit connections among them into a group of triples that explicitly referred to all those connections. As a result, we represented the triples as a group of tokens that the model could decode. In order to produce the correlations in the sentence as triples and reduce the number of tokens that must be



decoded, we developed a reversible linearization, incorporating special tokens. The data from the dataset served as the input for REBEL, and the output was the linearized triples. REBEL [27] is an autoregressive approach that frames Relation Extraction (RE) as a seq2seq [28] task. By pre-training an Encoder-Decoder Transformer (BART) [29] with our new dataset, REBEL achieved state-of-the-art performance on many RE baselines within a short time of fine-tuning. Its simplicity made it easy to adapt to new domains or longer documents. Since the same model weights were utilized after the pre-training phase, there was no need to train model-specific components from scratch. This made the training process more efficient. The aim of our model was to create  $y$  for a given  $x$ , if  $x$  was our input sentence and  $y$  was the outcome of linearizing the correlations in  $x$ . Utilizing cross-entropy loss in translation to fine-tune BART while performing a task, we optimized the log-likelihood of producing the linearized triples given the input.

### 4.3. Data preprocessing

The objective was to combine the results of the NLP phase (bi-LSTM and the triples generated by REBEL) into a single database and make necessary revisions. Each time a term was tagged and if the triple (term, subclass of, tag) was missing, it would be added to our database. In addition to this, we used the triples from the REBEL model which resulted in a coherent ontology that has a clear base structure. After conducting an analysis of the results of the REBEL model, we identified certain issues that required attention and subsequently resolved them before using the database to build the ontology. At first, the transformer generated two relations - *use* and *uses*, so we changed every relation *use* to *uses* in the database. Another problem was faulty triples, where one of them is empty, so those triples were deleted. Also, we had some terms that were the same except one was uppercase and the other was lowercase. As a solution, all uppercase letters were changed to lowercase to avoid having two classes representing the same concept. Some classes had multiple inheritances for one superclass causing an "Uncaught Exception in thread 'partitioning' " error. To resolve this, we deleted the first occurrence of *subclass of* that resulted in a cycle. This was an adhoc decision and would be evaluated further in the future. So the result of this part is our new database that contains various relations including *subclass of* relations which would be used next to build the ontology.

### 4.4. Implementation

#### 4.4.1. Reasoning

We had two approaches for reasoning. The two are slightly different but aim for the same results, which is to extract a machine-generated consistent graph and machine-generated inconsistent graph (both abbreviated mgg). And both used the well-known reasoner Konclude.

- First implementation: Our first implementation was to reason after the complete graph is created. So after we have our graph built, we reason on all the classes and delete each and every inconsistent class from our machine-generated graph to develop the machine-generated consistent ontology. Then we link those deleted classes to the GBO ontology to create the machine-generated inconsistent graph. The downsides or results of this approach are that some classes that are a subclass of an inconsistent class will be



flagged as inconsistent. Another downside is that some of the GBO classes can still be inconsistent. Thus another approach for the implementation was taken.

- Second implementation: Due to the downsides of the first approach, the second one was developed. Instead of reasoning after creating the graph, we reason while creating it. So after adding each class, we check with konclude, if a class / individual is consistent or not. If it's not consistent, then we delete it from our graph / ontology. But that in itself is not sufficient, since we also need to check after adding each relation if one of the axiomatically associated / linked classes became inconsistent. If it did become inconsistent, we removed it as well. Also if the class is from the GBO ontology, we removed the relation between the two classes, instead of deleting the GBO class itself. We have again generated the machine-generated consistent graph file and then we used the deleted classes to build the machine-generated inconsistent graph.

We used labels for synonyms and use natural language spaces, also because the IRI of a class isn't allowed to include symbols. Further adding labels with natural language spaces, improves the readability and is a best practice. The github repository of the implementation can be found here <sup>5</sup>.

#### **4.4.2. Human-machine interactive feedback loop**

This ontology graph is not our only result but also we have corrected the tags of our BI-LSTM using the inconsistent ontology file. So every class in the inconsistent ontology file was there because it had the wrong tags. So as a correction we did go through all those classes and then changed the tags in our csv file. This closes our loop. From tagging and creating the triplets to building our ontology to using this ontology to change the tags. By repeating this cycle, we can continuously enhance our outcomes.

#### **4.4.3. Closing the feedback loop with human interaction**

One method of closing our feedback loop is to use human interaction. The python script opens exactly in the position of the wrong tag and gives the user options to change the tag manually or to write or to type which tag should be in this position.

As we see in figure 4, at first the exact line 7 was opened then the feedback "hwp" (which is the tag for hardware part) was given so cpu's tag was changed to B-hwp. Then the exact line 35 was opened and the input "all cpu" was given. Then all the occurrences of cpu will be tagged as hwp. After this the script will go to the next inconsistent class until the user fixes all the incorrect tags. Since the current context is only microelectronic documents, we included that aid, as we expect wrong corrections rather improbable.

#### **4.4.4. Closing the feedback loop with machine automatic corrections**

Another way to close the loop is done by making machine automatic corrections. Not only is the human in the loop, but the results of the machine, e.g. the reasoning process are considered.

---

<sup>5</sup><https://github.com/OntologyResearcher/Supplementarymaterial>

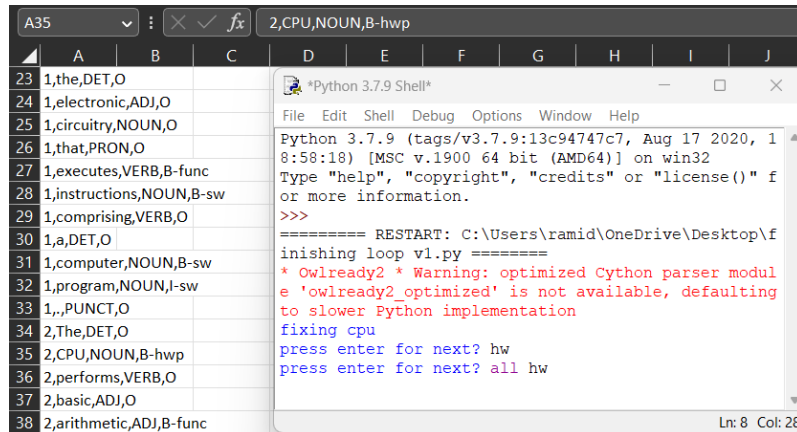


Figure 4: Example of human assisted correction

For example, the reasoner classifies something as inconsistent in the ontology, e.g. a hardware part that needs to be a system. The created inconsistency, which is the wrongly classified class as the tag from the dataset, will then first be removed. By having the loop the dataset is updated and as a consequence the predictions also change and improve, which again influences other tags.

## 5. Results

### 5.1. Resulting Knowledge Graph

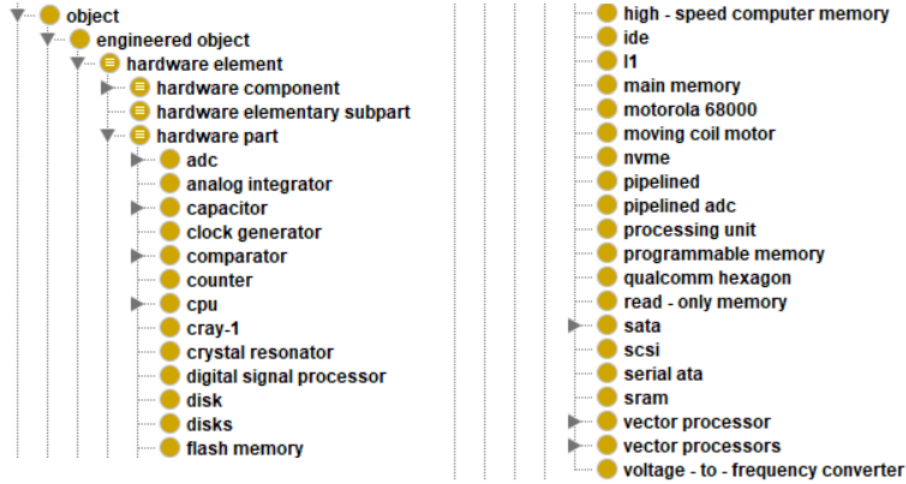
Metrics			
Axiom	17399	Class axioms	
Logical axiom count	8979	SubClassOf	8373
Declaration axioms count	3987	EquivalentClasses	10
Class count	3623	DisjointClasses	24
Object property count	93	GCI count	0
Data property count	3	Hidden GCI Count	6
Individual count	239		
Annotation Property count	35		

Figure 5: Graph metrics overview

We explore first the results of creating the ontology. We use protege to visualize the ontology, metrics are given in figure 5. The ontology has 3623 classes, 93 object properties, 239 individuals, 17399 axioms, and 8373 "subclass of" relations. Which is considerably larger than the manually developed models for specific use cases, that we were able to create in the past with considerable amounts of invested time and manual effort.

We already introduced an excerpt of the complete graph in Figure 2. Figure 6 shows the "hardware element" or hardware class from GBO. It has "hardware component","hardware

elementary subpart" ,"hardware part" and "hardware subpart" as subclasses. Here we show just the hardware part class of the inconsistent graph. We see that this class has many subclasses like adc, clock generator, cpu, crystal resonator, disks flash memory, serial ata and others. Which are often indeed hardware parts, but are inconsistent. Again those classes are machine-created and also machine classified as hardware part.



**Figure 6:** Inconsistent ontology OWL file overview with class "clock generator"

For example the class "clock generator" is inconsistent, because we have the rule that "hardware component", "hardware elementary subpart", "hardware part" and "hardware subpart" are disjoint classes. And the class clock generator is directly a subclass of "hardware component" and "hardware part." For this reason it is deleted from the main ontology and added to the inconsistent graph file.

## 5.2. Evaluation

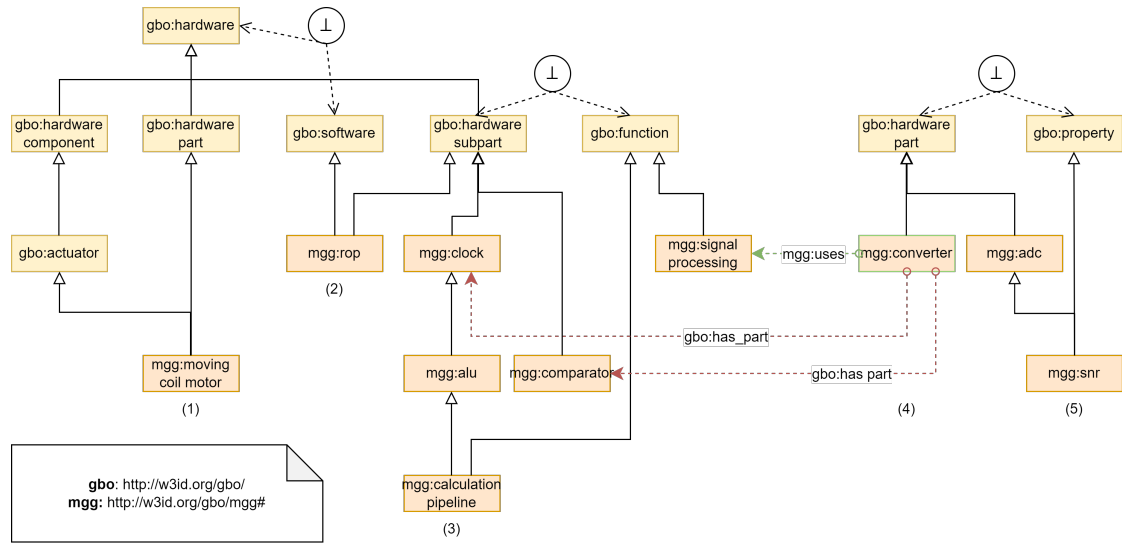
We evaluated the results according to the several metrics of conciseness and consistency. Conciseness refers to "avoiding schema and data elements irrelevant for the domain." (see footnote 3) Consistency measures the degree to which "(i.e., inconsistencies) with respect to the particular logical entailment considered." (see footnote 3) In order to handle complexity, we used only the transitive "has\_part" relationship for classification rather than the "has\_part\_directly" object property. This is equivalent with a less strict sorting out. First, we quantitatively measured the percentage of classes that the reasoner categorized. We counted the inconsistent and consistent classes with proteges metrics and subtracted the classes of both BFO and GBO. Whereas in the whole graph we did not subtract, as we see both ontologies as an integral element; unclassified classes were also counted directly. Table 2 gives an overview. Second, the amount of spared time by the knowledge engineer is significant. As the amount of data and its connections are hard to comprehend. This methods makes this process supervisable. Apart from the number of connections, some evaluations and considerations might simply be missed. The next paragraph gives examples. Third, we decided to qualitatively analyze and

**Table 2**  
Graph Partitioning after Reasoning

Graph Partition	Amount of Occurrence
unclassified classes	2218
unclassified individuals	239
GBO classes	83
inconsistent classes	165 (248-83)
inconsistent individuals	9
consistent classes	3399 (3482-83)
consistent individuals	194
complete graph classes	3623
complete graph individuals	239

evaluate the results and discuss their significance. Here, we exemplify the results based on examining the correctness through disjointness of the classes as well as the categorization through the "has\_parts" relationship. Figure 7 gives a sample of a few inconsistencies detected by the reasoner. In the following we will give explanations to each (1-5) case:

1. "moving coil motor" appears to be an unexpected classification by the neural net transformer, which puts it as a subclass of actuator (not hardware component) and contradicts our vocabulary.
2. "rop" appears at first glance to be an abbreviation of a hardware part. Examining the training dataset clears up the reason for the inconsistency. Rop is once classified as an abbreviation of "render output pipelines", a hardware part. And on another occurrence as an abbreviation of "return oriented programming", a software. Both terms are merged in our graph and thus create the inconsistency, because the abbreviation is referring to two different entities.
3. "calculation pipeline" was identifiably falsely assigned as a function in the tagging dataset by our trainee. The transformer in this case was correct by classifying it as a type of alu (arithmetic logic unit). In combination with our ontology the error was recognized.
4. here we see the "converter" class and its relationships. This class has as a part ("has\_part") a clock. For example, in digital-to-analog converters (DACs) and analog-to-digital converters (ADCs), a clock is often used to synchronize the conversion process. Also this class has as a part a comparator. In ADCs, the comparator is used to compare the input analog signal to a reference voltage and generate a digital output based on the result of the comparison. And this class uses some signal processing. In the context of converters, signal processing is used to transform the input signal into a format that can be processed and converted into the desired output signal. The color green indicates that the converter in itself is a consistent class. However, its relationships were removed, since both the comparator and the clock were found to be inconsistent classes.
5. "snr" (signal to noise ratio) was correctly tagged in the dataset as a quantity and thus correctly placed in the graph. The transformer neural network however, decided due to context to place the snr as a subclass of adc (as well as create the "facet\_of" relationship). An adc is a hardware part though, creating the inconsistency.



**Figure 7:** Examples for inconsistencies with disjointness axiom and "has\_part" object property

Overall the dataset was quantitatively improved by 4.7 % in coherency with inconsistency analysis alone. The question of relevancy is of similar importance to improve graph quality. The removing of unclassified classes achieved an improvement of 62,7 % in conciseness. Altogether our method improved our knowledge graph about 67,4 %. Table 3 summarizes the results.

**Table 3**  
 Quantification of Evaluation

Metric	Percentage of Improvement
unclassified removed classes	62,7%
inconsistent removed classes	4,7%
overall removed classes	67.4%

### 5.2.1. Discussion

This subsection puts the results into context and discusses them. Unclassified classes might be part of the correct graph but are just not classified, thus being falsely not classified. We always also have to weigh the NLP side of the classification when we want to assess a somewhat more complete or so called "absolute" correctness. It is important to put the results into context and discuss them to ascertain the meaning and understand the significance. For one we need to consider the current NLP accuracy of our transformer and training, which depends on the actual word of the vocabulary and is between 36-78 percent (F1 score, see [3]). These result depend on the dataset, how many samples have been used for training and depend on the word or tagging term. Other possible errors can be made by the transformer, the tagger and the quality or origin of the dataset. Things considered, sorting out with disjointness is a valuable addition to the construction process. Additional axioms and object properties (like "has\_parts") are

further expected to yield improvements. Coherency and succinctness are valuable construction metrics and it seems reasonable and preferable to follow the approach and dictum of quality over quantity.

## 6. Conclusion

This paper proposed a novel approach to building knowledge graphs with integrated reasoning. The approach sorts out inconsistent classes, relationships and instances and thus improves fitness for purpose of our knowledge graph in context of above discussion in a margin of up to 67,4 percent. By separating the graph into parts, the knowledge engineer is presented with only the erroneous parts, thus simplifies his work and makes it easier to oversee. As the approach is integrated with the dataset, the NLP accuracy is improved automatically by the knowledge engineers revision. The approach is demonstrated in the area of system engineering and electronics, but can easily be extended to other domains. To reapply the approach can already be done with a few disjoint classes that are used for training. Neural networks, such as transformers, large language models, Bi-LSTM or deep learning and machine learning can be applied in general. A more expressive ontology further increases the impact of our approach, as well as a systematic building approach with an upper ontology. Our work has various steps, but we made sure it is reusable for future engineers and people interested in using the approach. What is to be adapted is creating another ground ontology similar to GBO, that suits the aimed domain and tagging with other classes. Then the according automation can be used to finish generating the graph and recreate the mentioned feedback loop. This allows to create various ontologies and graphs in various domains with less inconsistencies. For example using a general "component" definition, which includes all of our parts. That can then also be matched accordingly. In future works we plan to incorporate more quality assurance aspects in automated ways and want to scale our approach. Rules, algorithms and reinforcement learning is further planned automation.

## References

- [1] J. Tang, Y. Yang, W. Wei, L. Shi, L. Su, S. Cheng, D. Yin, C. Huang, Graphgpt: Graph instruction tuning for large language models, 2023. [arXiv:2310.13023](https://arxiv.org/abs/2310.13023).
- [2] F. Wawrzik, A. Lober, A reasoner-challenging ontology from the microelectronics domain, in: G. Singh, R. Mutharaju, P. Kapanipathi (Eds.), Proceedings of the Semantic Reasoning Evaluation Challenge (SemREC 2021) co-located with the 20th International Semantic Web Conference (ISWC 2021), Virtual Event, October 27th, 2021, volume 3123 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 1–12. URL: <http://ceur-ws.org/Vol-3123/paper1.pdf>.
- [3] F. Wawrzik, K. A. Rafique, F. Rahman, C. Grimm, Ontology learning applications of knowledge base construction for microelectronic systems information, Information 14 (2023). URL: <https://www.mdpi.com/2078-2489/14/3/176>. doi:10.3390/info14030176.
- [4] M. C. Suárez-Figueroa, Neon methodology for building ontology networks: Specification, scheduling and reuse, 2010. URL: <https://oa.upm.es/3879/>. doi:10.20868/UPM.thesis.3879, ontology Engineering Group.

- [5] N. F. Noy, D. L. McGuinness, *Ontology Development 101: A Guide to Creating Your First Ontology*, Technical Report, 2001. URL: <http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html>.
- [6] A. Ayadi, A. Samet, F. de Bertrand de Beuvron, C. Zanni-Merk, *Ontology population with deep learning-based nlp: a case study on the biomolecular network ontology*, *Procedia Computer Science* 159 (2019) 572–581. URL: <https://www.sciencedirect.com/science/article/pii/S1877050919313961>. doi:<https://doi.org/10.1016/j.procs.2019.09.212>, *knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES2019*.
- [7] H. Alani, *Position paper: Ontology construction from online ontologies*, in: *Proceedings of the 15th International Conference on World Wide Web, WWW '06*, Association for Computing Machinery, New York, NY, USA, 2006, p. 491–495. URL: <https://doi.org/10.1145/1135777.1135849>. doi:10.1145/1135777.1135849.
- [8] A. Steigmiller, T. Liebig, B. Glimm, *Konclude: System description*, *Web Semantics: Science, Services and Agents on the World Wide Web* 27–28 (2014). doi:10.1016/j.websem.2014.06.003.
- [9] G. E. Trouli, A. Pappas, G. Troullinou, L. Koumakis, N. Papadakis, H. Kondylakis, *Summer: Structural summarization for rdf/s kgs*, *Algorithms* 16 (2023). URL: <https://www.mdpi.com/1999-4893/16/1/18>. doi:10.3390/a16010018.
- [10] M. Gao, J. Lu, F. Chen, *Medical knowledge graph completion based on word embeddings*, *Information* 13 (2022). URL: <https://www.mdpi.com/2078-2489/13/4/205>. doi:10.3390/info13040205.
- [11] J. Ma, C. Zhou, Y. Chen, Y. Wang, G. Hu, Y. Qiao, *Tecre: A novel temporal conflict resolution method based on temporal knowledge graph embedding*, *Information* 14 (2023). URL: <https://www.mdpi.com/2078-2489/14/3/155>. doi:10.3390/info14030155.
- [12] F. Al-Aswadi, H. Y. Chan, K. H. Gan, *Automatic ontology construction from text: a review from shallow to deep learning trend*, *Artificial Intelligence Review* 53 (2020) 1–28. doi:10.1007/s10462-019-09782-9.
- [13] S. Yuan, J. He, M. Wang, H. Zhou, Y. Ren, *A review for ontology construction from unstructured texts by using deep learning*, in: P. Kar, S. Guan (Eds.), *International Conference on Internet of Things and Machine Learning (IoTML 2021)*, volume 12174 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 2022, p. 121741D. doi:10.1117/12.2628713.
- [14] A. Schalkwijk, M. Yatsu, T. Morita, *An interactive virtual home navigation system based on home ontology and commonsense reasoning*, *Information* 13 (2022). URL: <https://www.mdpi.com/2078-2489/13/6/287>. doi:10.3390/info13060287.
- [15] B. Parsia, N. Matentzoglou, R. Gonçalves, B. Glimm, A. Steigmiller, *The owl reasoner evaluation (ore) 2015 competition report*, *Journal of Automated Reasoning* 59 (2017). doi:10.1007/s10817-017-9406-8.
- [16] X. Chen, S. Jia, Y. Xiang, *A review: Knowledge reasoning over knowledge graph*, *Expert Systems with Applications* 141 (2020) 112948. URL: <https://www.sciencedirect.com/science/article/pii/S0957417419306669>. doi:<https://doi.org/10.1016/j.eswa.2019.112948>.
- [17] R. Guimarães, A. Ozaki, *Reasoning in Knowledge Graphs*, in: C. Bourgaux, A. Ozaki,



- R. Peñaloza (Eds.), International Research School in Artificial Intelligence in Bergen (AIB 2022), volume 99 of *Open Access Series in Informatics (OASICS)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2022, pp. 2:1–2:31. URL: <https://drops.dagstuhl.de/opus/volltexte/2022/16000>. doi:10.4230/OASICS.AIB.2022.2.
- [18] M. Qu, J. Chen, L.-P. Khonneux, Y. Bengio, J. Tang, Rnnlogic: Learning logic rules for reasoning on knowledge graphs, 2021. [arXiv:2010.04029](https://arxiv.org/abs/2010.04029).
  - [19] M. Hildebrandt, J. A. Q. Serna, Y. Ma, M. Ringsquandl, M. Joblin, V. Tresp, Reasoning on knowledge graphs with debate dynamics, 2020. [arXiv:2001.00461](https://arxiv.org/abs/2001.00461).
  - [20] G. Wan, S. Pan, C. Gong, C. Zhou, G. Haffari, Reasoning like human: Hierarchical reinforcement learning for knowledge graph reasoning, in: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI’20, 2021.
  - [21] W. Chen, W. Xiong, X. Yan, W. Y. Wang, Variational knowledge graph reasoning, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), Association for Computational Linguistics, New Orleans, Louisiana, 2018, pp. 1823–1832. URL: <https://aclanthology.org/N18-1165>. doi:10.18653/v1/N18-1165.
  - [22] H. Ren\*, W. Hu\*, J. Leskovec, Query2box: Reasoning over knowledge graphs in vector space using box embeddings, in: International Conference on Learning Representations, 2020. URL: <https://openreview.net/forum?id=BJgr4kSFDS>.
  - [23] B. Xue, L. Zou, Knowledge graph quality management: A comprehensive survey, IEEE Transactions on Knowledge and Data Engineering 35 (2023) 4969–4988. doi:10.1109/TKDE.2022.3150080.
  - [24] R. Cohen, M. Geva, J. Berant, A. Globerson, Crawling the internal knowledge-base of language models, 2023. [arXiv:2301.12810](https://arxiv.org/abs/2301.12810).
  - [25] S. Dalecke, K. A. Rafique, A. Ratzke, C. Grimm, J. Koch, Sysmd: Towards “inclusive” systems engineering, in: 2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS), 2022, pp. 1–6. doi:10.1109/ICPS51978.2022.9816856.
  - [26] A. V. Peter Shaw, Jakob Uszkoreit, Self-attention with relative position representations, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies., Association for Computational Linguistics, 2017, p. 464–468.
  - [27] P.-L. Huguet Cabot, R. Navigli, REBEL: Relation extraction by end-to-end language generation, in: Findings of the Association for Computational Linguistics: EMNLP 2021, Association for Computational Linguistics, Punta Cana, Dominican Republic, 2021, pp. 2370–2381. URL: <https://aclanthology.org/2021.findings-emnlp.204>. doi:10.18653/v1/2021.findings-emnlp.204.
  - [28] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, volume 27, Curran Associates, Inc., 2014. URL: [https://proceedings.neurips.cc/paper\\_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf).
  - [29] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer, Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019. [arXiv:1910.13461](https://arxiv.org/abs/1910.13461).