A Reasoner-Challenging Ontology from the Microelectronics Domain

Frank P. Wawrzik[1], Andreas Lober[2]

[1]Technical University of Kaiserslautern, Erwin-Schrödinger-Straße 1, Kaiserslautern, 67663, Germany
[2]Technical University of Applied Sciences Ulm, Prittwitzstraße 10, Ulm, 89075, Germany

Abstract

This paper introduces a hardware software ontology from the microelectronics domain that suffers from the reasoning runtime bottleneck. We present it in detail, with a focus on the axioms related to reasoning. Further we explain the application in the GENIAL! project and make a runtime analysis of current reasoners on the ontology. We report a high runtime on a relatively small ontology, identify the reason and sketch (potential) solutions.

Keywords

reasoning, reasoner performance, GBO, ontologies, OWL, ISO26262, GENIAL!, Arrowhead Tools

1. Introduction

A 2020 study by Acatech[1] showed that the manufacturing industry is still lagging behind expectations in the design of systematic transformation programs. A survey by the Industry 4.0 Maturity Center came to the following conclusion: eighty percent of the companies surveyed have reached the second of six development stages toward Industry 4.0. The study shows a lack of standardization and a lack of implementation of technological concepts. One concept in this context is ontologies. Since real-world ontologies have a high degree of complexity, reasoners often fail here in live applications, making it impossible to work comfortably on the ontologies. Reasoning performance and tasks are significant bottlenecks for the breakthrough of semantic technologies and applications at a large scale. For ontologies that are applied and currently developed, for example for the IoT, reasoning is minimal or absent. In these cases, a domain description contains axioms mostly as a conceptualization of the domain, and that conceptualization then serves as a way to enable interoperability by opening up the data silos and serve as common semantic API. In this sense ontologies can also be useful without needing to use reasoning capabilities. An application is for example defining just the semantics to use them in e.g. triple stores. However, they then do not realize their potential and complete purpose.

SemREC'21: Semantic Reasoning Evaluation Challenge, ISWC'21, Oct 24 – 28, Albany, NY
wawrzik@cs.uni-kl.de (F. P. Wawrzik); lober@mail.hs-ulm.de (A. Lober)
https://cps.cs.uni-kl.de/mitarbeiter/frank-wawrzik-msc/ (F. P. Wawrzik);
https://www.uni-ulm.de/in/koop-promotionskolleg-cognitive-computing-in-socio-technical-systems/ueber-das-promotionskolleg/personen/andreas-lober/ (A. Lober)

[1]https://www.acatech.de/publikation/der-industrie-4-0-maturity-index-in-der-betrieblichen-anwendung/

This paper intends to contribute to the Reasoner Community by providing a real world needs ontology "in the hopes of directing reasoner developer attention to real user performance needs" (as stated in [1]). These cases are still rare, and are even less available online. And to support to reduce the hen and egg problem, mentioned in the SEMantic Reasoner Evaluation Challenge (SemREC) of 2021. Namely, that few real world ontologies are built, because reasoners are slow and hence, there are few examples to use to train and advance reasoner performance. We also seek solutions enabling the further building and realization of our application.
In this paper we describe an ontology for the microelectronics domain, called GBO (GENIAL! Basic Ontology) and a Hardware Software Domain Ontology, its parts related to reasoning and its application. Thus far, there has not been a consistent description of microelectronic systems, components, their context, properties, dependencies and functions. We introduce the current status of our ontology in detail. We explain how it is used in the context of our project and elaborate on the performance of the current reasoners. We further analyze possible improvements and workarounds. The paper is outlined as follows: In section 2 we give a brief summary of related ontologies and their reasoning aspects. In section 3, we extensively describe the ontology and its axioms related to reasoning. Further, we outline the profile and metrics of the ontology. In section 4 we show and evaluate the performance for a starting point of the evaluation challenge and depict solutions. Section 5 concludes and mentions future work.

2. State of the art

A wide variety of reasoners can be used for the process of reasoning when checking and merging ontologies—the three best known and most widely used reasoners for protégé are FaCT++, Pellet, and HermiT and are used in this paper for evaluation.
In a 2006 paper by Tsarkov and Horrocks [2], FaCT++ was presented as a description logic reasoner that implemented a Tableaux decision procedure for the well-known SHOIQ description logic. The reasoner has a standard DIG interface so that reasoning services can be provided to ontology engineering tools. The focus of this reasoner is on reducing the number of subsumption tests during the classification.
Sirin [3] presented the reasoner Pellet in 2007, which, on the other hand, is characterized by its complete reasoning. The reasoner is called the complete OWL-DL reasoner by the authors. Pellet was created with most of the state-of-the-art optimization techniques provided in the DL literature. These functions include normalization, simplification, absorption, semantic branching, back jumping, caching satisfaction status, top-bottom search for classification, and model merging.
The reasoner HermiT was developed by Glimm and introduced in 2014 as a new type of reasoner fully compliant with OWL 2 Direct Semantics [4]. This reasoner is based on the Hypertebleau calculus, supports standards and optimizations that improve the performance of ontology inference. HermiT also supports several functions outside of the OWL 2 standard. Examples of this are SPARQL queries and description graphs.
In a recent paper Glimm [5] uses parallelisation and an individual derivation cache approach to significantly improve ABox reasoning performance. Recent advances in machine learning foster their application to description logic-based reasoners. A recent contribution by Singh

and Mutharaju[6] that combines an ontology-based embedding model with reinforcement learning shows promising results. They are further working on combining approximation and concurrency to increase runtime for dedicated purposes and on distributed reasoning. In [7] the authors made an analysis on why the reasoning is so costly, applied machine learning techniques and could outperform existing reasoners.
Digital Reference (DR)[2] [8] is a neighbouring ontology describing terms for the semiconductor supply chain. It serves rather as a light structured vocabulary with fewer, but some axioms used for reasoning (often equivalent classes). It contains a larger axiom count (around 7,000 in 2021) than this ontology and had a reasoning time of 15 seconds with pellet full reasoning on the MacBook Air machine that is used in the evaluation in section 4. In comparison with Digital Reference, our basic ontology intended to capture the domain of hardware and its parts with a consistent domain description and application of consistency checking of definitions, whereas DR is rather a reference for relevant terms.
A major development is the deployment of ontologies in IoT infrastructures in the context of the Horizon2020 programs. One such project VICINITY [9] enabled bottom-up interoperability as a service in the internet of things domain for smart neighbourhoods. Its ontology[3][4] has significant overlap with this ontology and has slight differences in conceptualization (e.g. System, Device, PhysicalThing and Sensor). However, reasoning did not play a significant role.
The ontology of units of measure and related concepts[10] consumes a high amount of reasoning time, which made us not import this ontology. Checking for right units allocation for example seems a useful benefit, but since it was not practical, we just adopted the conceptualization of quantity, measure and unit to consistently describe our properties and constraints.
The SSN ontology [11] is a small ontology that does not suffer from long reasoning. In this regard, the ontology in this paper is one of the few examples for a reasoning bottleneck in our domain.
Our ontology includes a top level approach with some mediating middle-level classes and is a good candidate for further integration of neighbouring ontologies. Recent developments in modularization approaches that can lessen load on reasoning are for example MODDALS [12]. The identification of knowledge areas and their proper partitioning into common domain, domain-variant, domain-task and application layer supports a high degree of reuse and use ability.

3. Hardware Software Domain Ontology

3.1. Application

3.1.1. In Context of the Projects

In the context of the GENIAL! project, GBO was introduced to facilitate the development of microelectronic components (and innovations related to automotive and electronics in general) across the automotive value chain. We create a tool that helps the value chain coordinate, evalu-

[2] http://www.w3id.org/ecsel-dr
[3] http://vicinity.iot.linkeddata.es/vicinity/
[4] http://iot.linkeddata.es/def/core/index-en.html

ate and explore architectures, performances, and innovations. These functions are accomplished by an exchange of models and interdependencies throughout OEM, Tier1, and Tier2.The tool checks the constraints, does consistency checking in the form of formulas, and then allows to further make estimations based on roadmap defined predictions. We formulate constraints as boolean or arithmetic formulas described in a self developed language called APPEL [13]. This allows (1) constraints to be linked to the knowledge base and (2) easy to understand and use descriptions and (3) a practical way to explore changing constraints and ranges. The tool consists of

- Frontend (to interactively explore the model and its constraints, visualizations, user management)
- Backend (ArangoDB database intertwined with OWL and APPEL)
- jAADD library (For symbolic computation of constraints, which saves all dependencies among variables and constraints)

For doing the exchange, we also need a consistent description of our models that is common to all participants along the value chain. In this sense, GBO is imported as top level across a distributed architecture for every participant using our tool. This supports computer assisted tasks, like for example the application of rules, the exchange of models or simply good data quality.
In the Arrowhead Tools project, we create system-of-systems (SoS) interoperability along with the SoS toolchain. Targeting for the first time tools, rather than devices or other specific domain descriptions. There, candidate ontologies are currently evaluated in the Industry 4.0 domain and ours helps integrate the domains. The ontologies are linked with other languages like SysML to integrate the SoS domain, like in [14].

3.1.2. In Context of Ontologies

GBO[5] was designed to yield a consistent description based on a well-established standard called ISO26262. It is a standard in safety to describe related terms in the context of automotive and functional safety of microelectronics. Moreover, the Hardware Software Ontology contains the vocabulary that GBO controls with its axioms. When there is not a reasoner error, all terms are adhered to and they are correctly classified through the standard (because we often use strict universal restrictions).
In the project, we develop several ontologies in microelectronics. Usually we refer to their application in car innovation and they are related to roadmap models, that for example include battery density and price predictions. Further a neural net accelerator knowledge base we use, also has increased reasoning time by applying transitive and inverse hasParts relations, but is just yet acceptable in runtime. The repositories to download the ontologies can be found on github with a saved status for the reasoner challenge and an introduction of GBO can be found on a website. The links are provided in the appendix of the online resources at the end of the paper.

[5] http://w3id.org/gbo

3.2. GENIAL! Basic Ontology

The GENIAL! Basic Ontology is based on BFO (Basic Formal Ontology). BFO is a widely used top-level ontology that is meant to facilitate interoperability between ontologies. It contains classes to model for example objects, functions and qualities. Its basic definitions supported a clearer classification and structuring of basic terms in our ontology and its basic axioms support reasoning with disjoint axioms. The upper part of figure 1 shows the BFO part of the continuant in GBO. Some basic terms in GBO that for example are used to define a GBO controller and that can also be found in figure 1, among others, are: *element, non system level element, system level element, function, property, software element (software), hardware element (hardware), processor, system, hardware part, software unit, hardware elementary subpart, hardware subpart, unit, measure, quantity, sensor, actuator, operating time, component, hardware component, software component, controller, dependency* and *context. Hardware component* and *hardware part* for example are connected with a hierarchical has part directly relationship and a cardinality restriction (*hardware part* is the next hierarchical level). We use has part directly as non-transitive in order to reason with existential restrictions. Defining an existential restriction with the transitive has part produces an error. The *function* term is defined twice, once in BFO and once again more precisely in GBO for our scope. All terms in GBO are labeled terms and Webprotege just displays some with underscore and some not.

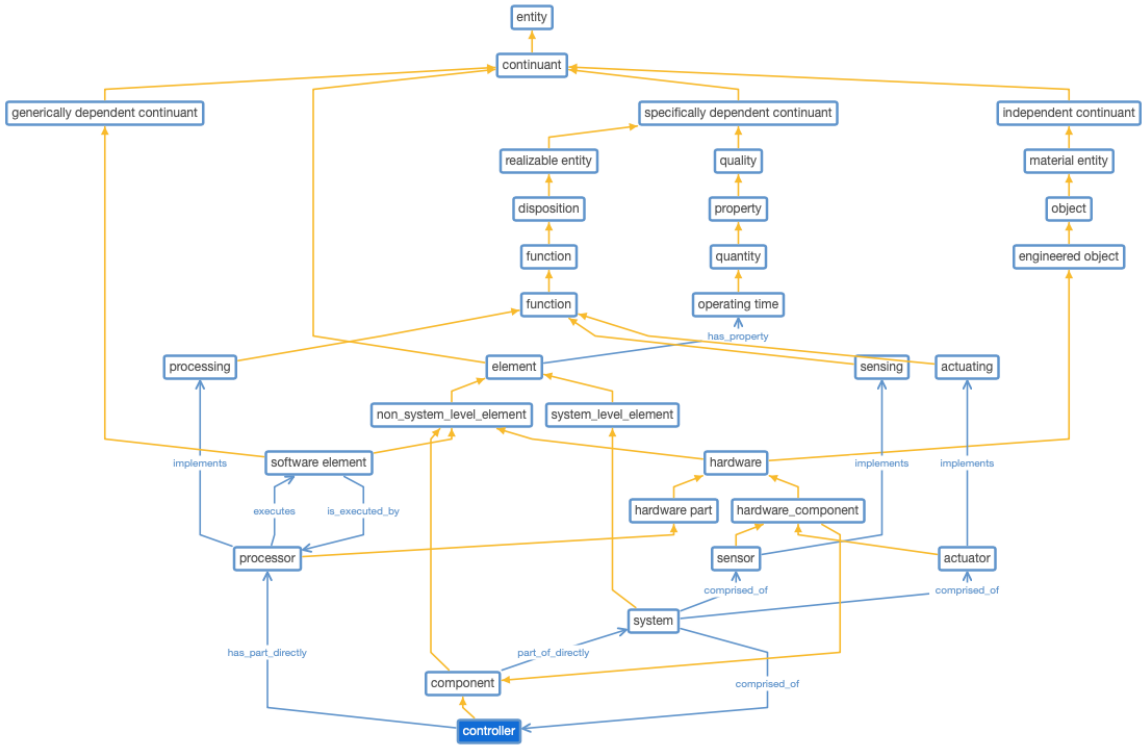


Figure 1: Explicit Definition of a GBO Controller with its Superclasses and Relationships

The ISO terms were formalized according to their definitions. It structures the hardware (and software) in five (three) stratified decomposition levels from system to hardware elementary subpart. Terms are linked with object properties:

- has part directly / part of directly (elements and others), has part / part of (elements and others)
- implements / is implemented by (functions), executes / is executed by (HW / SW)
- has property (properties), depends on (dependencies, constraints)
- has value (measures, concrete values), has unit (for units)

3.2.1. Reasoning Axioms

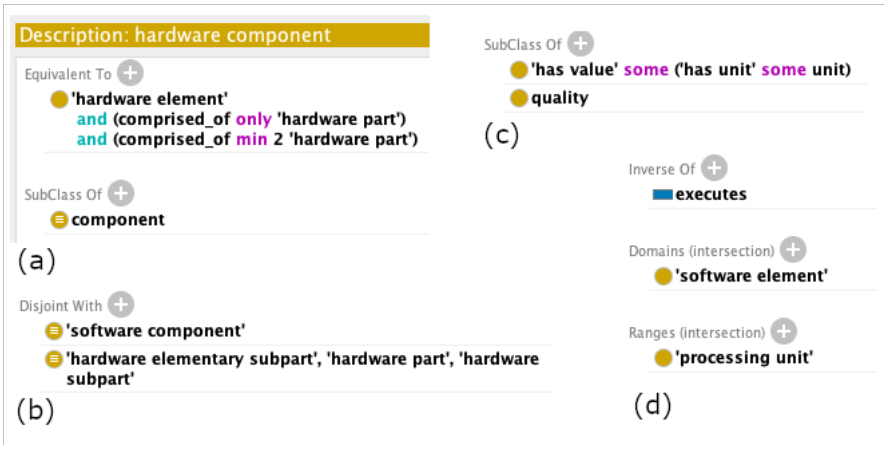


Figure 2: Small Collection of GBO Axioms

We used a breadth of reasoning constructs together in this ontology, when seen as appropriate to use. Some of which are illustrated in the following:

- Equivalent Classes, Existential and Universal Restrictions, Cardinality Restrictions, Logical Operations (Figure 2 a)
- Covering Axioms, Disjointedness (Figure 2 b), Domain and Range Axioms (Figure 2 d)
- Nested Restrictions (Figure 2 c), Transitivity, Symmetry and Inverse Relations (Figure 2 d)

Figure 2 a shows the definition of hardware component, figure 2 b disjointedness of hardware component, figure 2 c the property definition and figure 2 d the is executed by definition.

In GBO, we did not make use of property chains and rules. It was observed that the definitions provided by ISO26262 were well organised and included knowledge of ontology principles. Thus far, it did not create problems with the reasoner (in terms of incorrect definitions and conceptualizations), when building the library at this point. Although the creators certainly did not create an OWL file and check this against real definitions in a formal model.

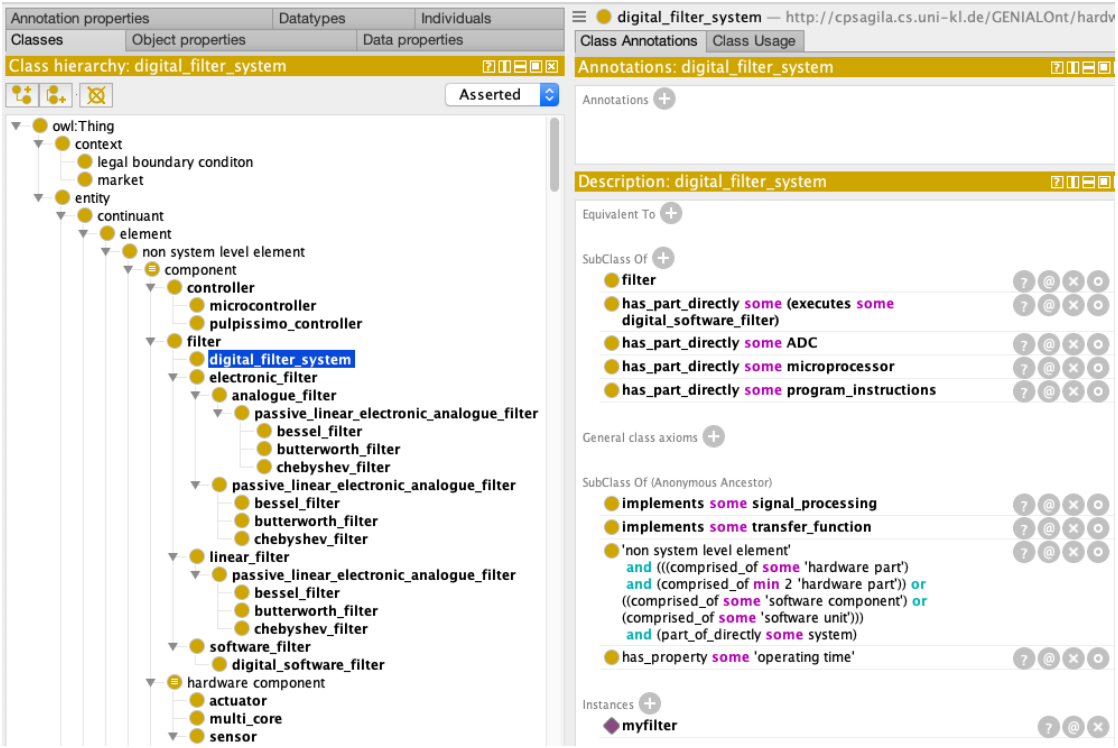


Figure 3: Excerpt from the Filter Domain from Hardware Ontology

3.3. Microelectronics Domain Ontology

So far the ontology consists of

- a neural net accelerator use case from processor experts. It describes its components, functions and its properties and allocates software components or software units to the processor on which they are executed
- components of a simulation library of cyber-physical systems (with a years long background in high level HW/SW co-design, power profiling and verification)
- wikipedia and expert articles

Which we then divide into application specific and common terms. Figure 3 shows an excerpt of the hardware domain with a few filter classes. It can be noted, that the filter system here, actually is a component which is disjoint to a system. But since that is an ISO definition with a namespace, there is no contraction. It just shows, that different parties use different words for the same thing. The figure illustrates the use of the has parts, implements and has property relations to yield the formal definitions of our parts. GBO is inherited and controls, if the digital filter system really is a component. By for example requiring, that an analog to digital converter (ADC) is a hardware part. Also a component is something which consists of both, hardware and software and is part of a system.

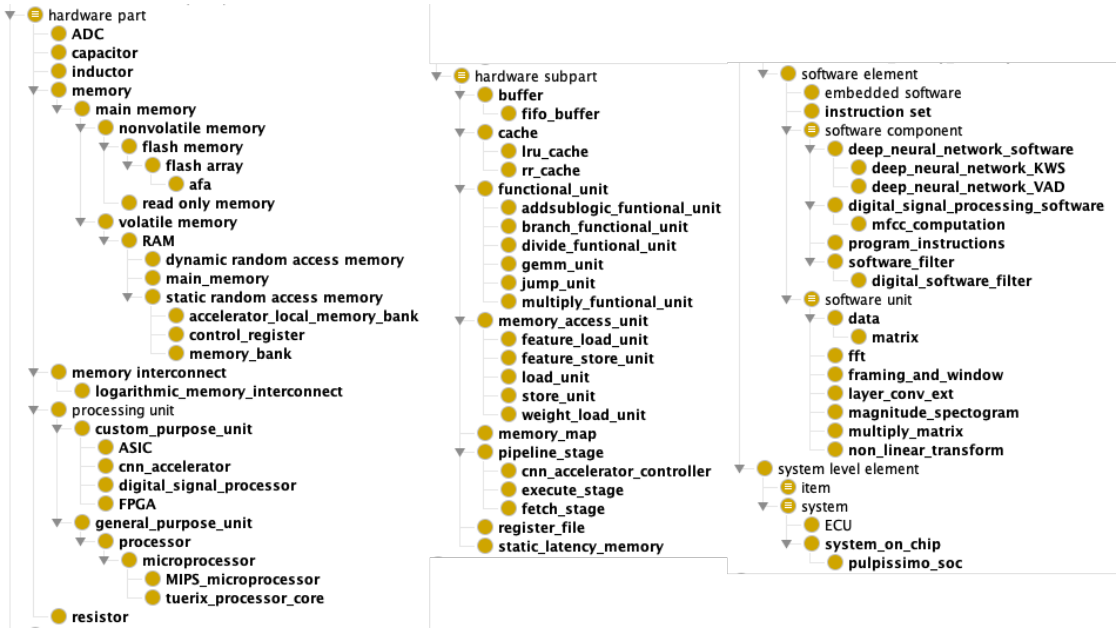
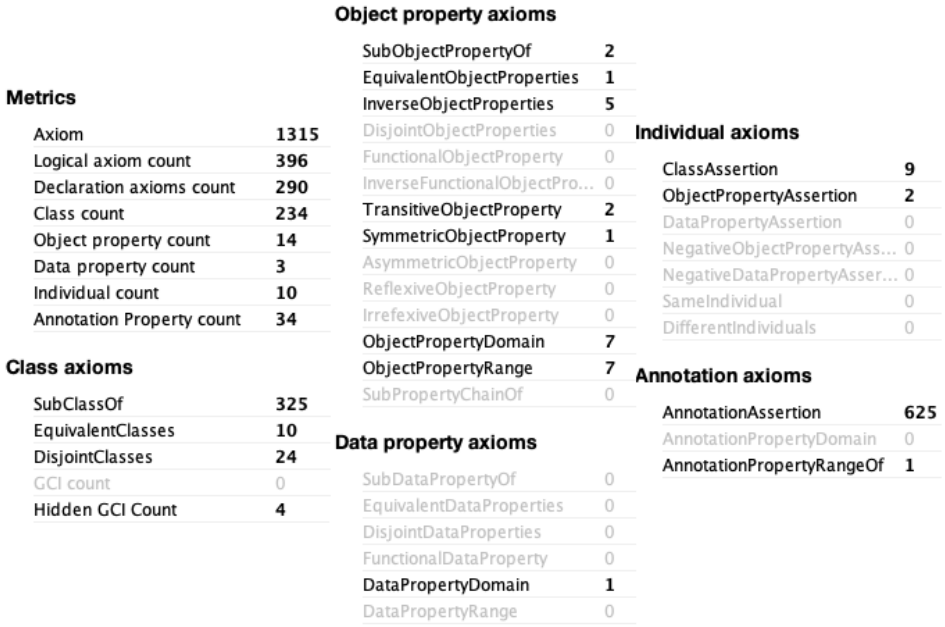


Figure 4: Excerpt from Hardware Parts, Hardware Subparts and Software from HW / SW Ontology

Figure 4 shows further classification of hardware parts, hardware subparts, and software elements, in order to illustrate parts of the ontology. Functions, properties and more parts are omitted due to reasons of space. Sometimes definitions and definition sources are provided. But already the size is large enough so that the ontology is difficult to be maintained for a single person. Reasoning axioms were kept minimal for the neural net use case. And the classifications are just basic enough for working with a handful of hardware experts and for one use case. We see various memories, processors, electronic components, functional units, memory access units and more parts that are relevant for processors. The software components are used for keyword spotting and voice activity control of the neural net on a cnn accelerator.

3.4. Ontology Profile

Figure 5 shows the metrics and profile of our ontology. According to official classifications, it is a rather small to medium ontology. It was designed in a minimal fashion, with few object properties and data properties. We made more use of appropriate annotations, especially in GBO, in order to give usage examples, references, the ISO definitions and labels. And tried to avoid axioms that do not yield actual benefit in application.



Metrics

Axiom	1315
Logical axiom count	396
Declaration axioms count	290
Class count	234
Object property count	14
Data property count	3
Individual count	10
Annotation Property count	34

Class axioms

SubClassOf	325
EquivalentClasses	10
DisjointClasses	24
GCI count	0
Hidden GCI Count	4

Object property axioms

SubObjectPropertyOf	2
EquivalentObjectProperties	1
InverseObjectProperties	5
DisjointObjectProperties	0
FunctionalObjectProperty	0
InverseFunctionalObjectPro...	0
TransitiveObjectProperty	2
SymmetricObjectProperty	1
AsymmetricObjectProperty	0
ReflexiveObjectProperty	0
IrrefexiveObjectProperty	0
ObjectPropertyDomain	7
ObjectPropertyRange	7
SubPropertyChainOf	0

Data property axioms

SubDataPropertyOf	0
EquivalentDataProperties	0
DisjointDataProperties	0
FunctionalDataProperty	0
DataPropertyDomain	1
DataPropertyRange	0

Individual axioms

ClassAssertion	9
ObjectPropertyAssertion	2
DataPropertyAssertion	0
NegativeObjectPropertyAss...	0
NegativeDataPropertyAsser...	0
SameIndividual	0
DifferentIndividuals	0

Annotation axioms

AnnotationAssertion	625
AnnotationPropertyDomain	0
AnnotationPropertyRangeOf	1

Figure 5: Ontology Profile and Metrics of HW / SW Ontology

4. Reasoning with the Ontology

4.1. Performance

Compared to other ontologies, the reasoning runtime was quite high for the size of the ontology. Table 1 shows the machines on which we tested the ontology, as well as the runtime of Pellet, HermiT, and FaCT. The only reasoner that had a more or less acceptable runtime was FaCT. But still too slow when further axioms are added. Other reasoners (jcel, ontop, ELK) were not able to handle the ontology at all. Likely because of its expressiveness. The HermiT reasoner performed better than Pellet, but it is still too slow to enable active work with a running reasoner on the ontology.

Table 1 clearly shows that commercially available computers, which can also be found in the work-related environment of the industry, have problems with the ontology. The evaluations showed that even computers with high computing power have problems running the reasoners, and none of the reasoners can show a satisfactory runtime. Even the well-equipped Fujitsu ESPRIMO P956 showed a very high CPU utilization of 90 percentage at times during the execution of the Pellet Reasoner, whereby the utilization was at 16 percentage most of the time. Memory usage was not maxed out. On average, the CPU utilization was between 16-60 percentage for all computers and the Reasoner application. The table shows that computing power and memory alone are not the critical factor, but it hints at other bottlenecks.

Table 1
Machines and Reasoner Performance

Machine Configuration	Reasoner Runtime[a]		
	Pellet	*HermiT 1.4.3.456*	*FaCT++ 1.6.5*
MacBook Air, 8GB Memory Intel i7 1,7 GHz, 2 cores, 4MB L3-Cache	>15 min	3 min 50 sec	20 sec - 1 min 15 sec
Fujitsu Lifebook, 16GB Memory Intel i5 1,7 GHz, 4 cores, 6MB L3-Cache	>15 min	2 min 33 sec	40 sec - 1 min 20 sec
Lenovo Y50, 8GB Memory Intel i7 2,6 GHz, 4 cores, 6MB L3-Cache	>15 min	8 min - 13 min 10 sec	2 min 40 sec - 3 min 10 sec
Lenovo LNVNB161216, 8GB Memory AMD A9 2,9 GHz, 2 cores, 2MB L2-Cache	>15 min	>15 min	2 min 26 sec
Fujitsu ESPRIMO P956, 32GB Memory Intel i7 3,4 GHz, 4 cores, 8MB L3-Cache	>15 min	2 min 30 sec	40 sec

[a]Full Reasoning/Inference.

4.2. Approaches for Performance Resolutions

4.2.1. Modularization

To simplify the reasoning, we thought of modularizing each part. But, if a certain specialization of a filter system is having a specific ADC, both ontologies (of filters and ADC types) must be imported for one specific reasoning task. Which might proof difficult for large scale usage and if ontologies are very intertwined.

4.2.2. Outsourcing of Reasoning-Process

Another performance resolution could be outsourcing the reasoning process to a data center that could perform the reasoning via a web service even with more complex ontologies. However, the question arises, if industries are willing to map their sensitive data in an ontology to be processed on another server. Such a concept can also be suitable for supply networks within an industry through a merger of OEMs and 1-tier suppliers.

4.2.3. Removing Axioms

Removing the definition of component resolved the reasoning time for HermiT and FaCT. Hermit (4 sec) even outperformed FaCT (6 sec) on the first machine after this change. However Pellet was still suffering from long reasoning (aborted after 3 min). The definition of component did not cause reasoning issues for just GBO. Nesting the definition with brackets or removing equivalent object properties (comprised of equivalent to has parts directly) also was not the problem and the formal definition was correct. In fact, the analysis showed that the axiom ”component part of directly some system” (see figure 1) was the reasoning issue. We can consider,

although contained in the ISO definition and correct, abstaining from using the axiom in this case, since the reasoning time is not worth it (which is suboptimal). But we also suggest that in theory handling this axiom can be improved by future reasoners.

5. Conclusion and Future Work

The authors see five major bottlenecks for the advancement of semantic web technologies. Those are (1) reasoner performance has to be improved in the order of magnitudes. (2) automatic construction of knowledge bases and harnessing the amount of information/data we have. (3) the willingness and approaches to share ones own and interact with others data. (4) complexity management and (5) quality of ontologies and their unification. This paper faced the challenge of point (1). It presented the GENIAL! Basic Ontology (GBO) to describe basic parts of hardware and software and related information. GBO was applied to a HW / SW Ontology Library that created reasoning issues. We outlined some of its axioms in detail. Finally, we elaborated on the performance runtime of some well-known and prevalent reasoners on our ontologies.
In future works, we will further look to workarounds and build reasoner test cases that allow to check, if we actually modelled what we wanted to with a more consistent methodology and library. Instead of just using it for consistency checking of our definitions, we want to infer new knowledge and apply rules from expert systems, which will further strain the reasoner.

Acknowledgments

This work has been supported by the GENIAL! project with funding from the BMBF under grant agreement No 16ES0865K. This work also has received funding from the EU ECSEL Joint Undertaking under grant agreement no. 826452 (project Arrowhead Tools) and from the partners national funding authorities BMBF under the no. 16ESE0359.

A. Online Resources

The sources from this paper are available via

- GitHub Repository of the Project.
- GitHub Repository for SemREC 2021 Status.
- GBO Overview.

References

[1] B. Parsia, N. Matentzoglu, 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.

[2] D. Tsarkov, I. Horrocks, Fact++ description logic reasoner: System description, in: U. Furbach, N. Shankar (Eds.), Automated Reasoning, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 292–297.

[3] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical owl-dl reasoner, Journal of Web Semantics 5 (2007) 51–53. URL: https://www.sciencedirect.com/science/article/pii/S1570826807000169. doi:https://doi.org/10.1016/j.websem.2007.03.004, software Engineering and the Semantic Web.

[4] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, Hermit: An owl 2 reasoner, Journal of Automated Reasoning 53 (2014). doi:10.1007/s10817-014-9305-1.

[5] A. Steigmiller, B. Glimm, Parallelised abox reasoning and query answering with expressive description logics, in: R. Verborgh, K. Hose, H. Paulheim, P. Champin, M. Maleshkova, Ó. Corcho, P. Ristoski, M. Alam (Eds.), The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings, volume 12731 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 23–39. URL: https://doi.org/10.1007/978-3-030-77385-4_2. doi:10.1007/978-3-030-77385-4_2.

[6] G. Singh, S. Mondal, S. Bhatia, R. Mutharaju, Neuro-symbolic techniques for description logic reasoning (student abstract), Proceedings of the AAAI Conference on Artificial Intelligence 35 (2021) 15891–15892. URL: https://ojs.aaai.org/index.php/AAAI/article/view/17942.

[7] Y. B. Kang, S. Krishnaswamy, W. Sawangphol, L. Gao, Y.-F. Li, Understanding and improving ontology reasoning efficiency through learning and ranking, Information Systems 87 (2019). doi:10.1016/j.is.2019.07.002.

[8] H. Ehm, N. Ramzy, P. Moder, C. Summerer, S. Fetz, C. Neau, Digital reference – a semantic web for semiconductor manufacturing and supply chains containing semiconductors, 2019, pp. 2409–2418. doi:10.1109/WSC40007.2019.9004831.

[9] Y. Guan, J. Vasquez, J. Guerrero, N. Samovich, S. Vanya, V. Oravec, R. García-Castro, F. Serena, M. Poveda-Villalón, C. Radojicic, C. Heinz, C. Grimm, A. Tryferidis, D. Tzovaras, K. Dickerson, M. Paralic, M. Skokan, T. Sabol, An open virtual neighbourhood network to connect iot infrastructures and smart objects — vicinity: Iot enables interoperability as a service, 2017 Global Internet of Things Summit (GIoTS) (2017) 1–6.

[10] H. Rijgersberg, M. Assem, J. Top, Ontology of units of measure and related concepts, Semantic Web 4 (2013) 3–13. doi:10.3233/SW-2012-0069.

[11] M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. Le Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, K. Taylor, The ssn ontology of the w3c semantic sensor network incubator group, Web Semant. 17 (2012) 25–32.

[12] J. Ariza, F. Larrinaga, E. Curry, Moddals methodology for designing layered ontology structures, Applied ontology Pre-press (2020). doi:10.3233/AO-200225.

[13] C. G. Frank Wawrzik, Alexander Louis-Ferdinand Jung, Appel - agila property and dependency description language, in: MBMV 2021 - 24. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, 2021.

[14] G. Kulcsar, K. Koltai, S. Tanyi, B. Peceli, A. Horvath, Z. Micskei, P. Varga, From models to management and back: Towards a system-of-systems engineering toolchain, 2020, pp. 1–6. doi:10.1109/NOMS47738.2020.9110310.