

Copyright © 2018 for this paper by its authors. Copying permitted for private and academic purposes

## A Pattern-Based Core Ontology for Product Lifecycle Management based on DUL

Falko Schönsteich<sup>1</sup>, Andreas Kasten<sup>2</sup>, Ansgar Scherp<sup>3</sup>

<sup>1</sup> Christian-Albrechts-Universität, Kiel, Germany

[falko.schoenteich@stu.uni-kiel.de](mailto:falko.schoenteich@stu.uni-kiel.de)

<sup>2</sup> Debeka, Koblenz, Germany

[andreas.kasten@debeka.de](mailto:andreas.kasten@debeka.de)

<sup>3</sup> University of Stirling, Scotland, UK

[ansgar.scherp@stir.ac.uk](mailto:ansgar.scherp@stir.ac.uk)

**Abstract.** A major challenge in Product Lifecycle Management (PLM) is the exchange of product data across lifecycle phases, information systems, and parties as data formats, structure and quality can vary vastly. Existing approaches focus only on certain phases of PLM, predominantly design and manufacturing, while the subsequent phases of usage/maintenance and disposal are often ignored. However, especially the usage phase is becoming increasingly important for revenue as customer expectation for services beyond the initial purchase of a product is growing. This paper proposes an ontology CO-PLM based on the foundational ontology DOLCE+DnS Ultralite to provide a formal basis for PLM. In contrast to existing approaches, CO-PLM follows an holistic approach covering all phases of PLM and integrates patterns from existing core ontologies.

**Keywords:** product lifecycle management, pattern-based ontologies

### 1 Introduction

Product Lifecycle Management (PLM) sums up all activities regarding the design, production, usage, and termination of products. It includes the processes from the first ideas up to disposal [1]. A central challenge of PLM is sharing information and knowledge between different organizational parties, such as departments, companies, etc., over the whole lifecycle of a product [2]. This is not a trivial task, as many different software systems used by companies today to support their business processes often rely on individual data models and lack interoperability with one another. Several models exist for dividing a product's lifecycle in separate phases. In general, these consist of four main phases describing the early, middle, and late life of a product. Depending on the specific product, the phases may involve different business processes. In context of manufacturing, the **design phase** consists of requirement engineering, product specification, and prototyping. This phase starts with the very first idea of a new product. Most of research and product development takes place in this phase. Next, the **production phase**

describes the processes around procurement and disposition of the product's components as well as its assembly. In the **usage phase** the product is sold to the customers. Most of spare parts management and customer service management takes place in this phase. The product lifecycle ends with the **disposal phase**. The product gets either replaced, recycled, or eliminated, when it is not of sufficient use any more. The activities in all phases may differ depending on the components of the product. Most high-tech products consist of multiple types of components, e. g., electrical, electronic, mechanical, hydraulic/pneumatic, or software. For each of these types certain pieces of information are created and have to be managed. This process can be digitally supported by Product Data Management (PDM) software systems. The most common data of a high-tech product's lifecycle to be managed are its metadata, e. g., identification numbers, spatial dimensions, weight, or storage conditions. Depending on the industry, further artifacts are relevant, e. g., technical specifications, geometrical drawings, and 3D-models usually summarized as Computer Aided Design (CAD) data, manuals, and test reports.

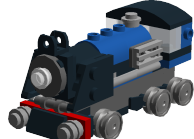
Several ontologies covering concepts of PLM already exist and can be categorized into three groups: engineering [3,4,5], manufacturing [6,4,5], and lifecycle ontologies [7,8]. **Engineering-centred ontologies** classify product features, e. g., drilling holes or edges, while **manufacturing-centred ontologies** focus on describing resources and processes on the shop floor level, e. g., machine, schedule, and production steps. In contrast to these, **lifecycle-centred** ontologies try to describe product lifecycle management comprehensively by focusing on more general concepts. They may also refer to products and machines, but not as explicitly as ontologies of the other two types. They rather focus on activities connecting resources and information from different lifecycle phases. In comparison to engineering and manufacturing centred ontologies, lifecycle-centred ontologies address also a more abstract meta-level of Product Data Management.

Most literature and software solutions for PLM focus on the design and manufacturing phases, as these two are considered to have the most business impact [9]. **The CO-PLM model presented in this paper** shares this view, but **extends the state of the art by supporting also the remaining phases, especially the usage phase.**

Section 2 contains a more detailed problem description for PLM using a LEGO model as an illustration. In Section 3, we describe the requirements for a core ontology for PLM. As none of the ontologies in the related work cover these entirely, we present the pattern-based design of CO-PLM in Section 4. Section 5 investigates how the use of CO-PLM scales with increasing number of product parts in terms of axiom count and reasoning time, before we conclude the paper.

## 2 Problem Description

We illustrate the problem using a commonly known example of a LEGO model. It serves as illustration of several PLM concepts and challenges, i. e., part identification, subpart relations or assembly processes. Figure 1 shows LEGO Set 31504







Element-ID	Name	Picture	Design-ID	Color code	Quantity
4211098	BRICK 1X1		3005	199 - Dark Stone Grey	2
306540	BRICK 1X2 WITHOUT PIN		3065	40 - Transparent	2
4211060	BRICK 2X2		3003	199 - Dark Stone Grey	1
4583862	BRICK 1X1 W. 1 KNOB		87087	23 - Bright Blue	4

Fig. 1: Blue Express Train with Bill of Materials (extract)

“Blue Express” and an extract of its Bill of Materials (BOM). The model consists of 71 individual bricks of 32 different brick types. A BOM “lists the subassemblies, intermediate assemblies, component parts, raw materials, and quantities of each needed to produce one final product” [10]. Therefore, the BOM can be different for the design phase, called Engineering Bill of Materials (EBOM), in comparison to the production phase’s Manufacturing Bill Of Materials (MBOM). The shown BOM is generated by the LEGO Digital Designer. The most common attributes of BOMs are an ID and the amount. Other common content is the color, pictures of the parts, materials or units, especially when liquids, gases, or bulk stock is used. The following sections illustrate differences between EBOM and MBOM.

## 2.1 Engineering and Manufacturing View on Parts

LEGO bricks have two different IDs which are Element IDs and Design IDs. The Design ID defines the geometry of a LEGO brick. For example, a two by two standard brick will always have the same Design ID, regardless of color or graphics printed on it (Figure 2a, left side). The Element ID on the other hand refers to a specific LEGO brick, including its Design ID, color, printing, etc., which can be bought from the parts shop, or is referenced in a building manual (Figure 2a, right side). This distinction illustrates different views on the same part in a real life scenario. An engineer is more interested in form, fit, and function of a product or a part, while parts in manufacturing must be identifiable in an unambiguous way (e. g., for retrieval from a storage system). Therefore the engineer only needs to specify the design of a part as it is not necessary to dictate, e. g., the color of the part. Besides color, the supplier may be another discriminator to have several (Element) IDs for the same functional part (Design ID). In general, the “engineering version” only depicts form, fit, and function while the “manufacturing version” adds, e. g., color, printing, or supplier. This separation implies that one engineering part may refer to many manufacturing parts, while usually one manufacturing part refers to exactly one engineering part, as depicted in Figure 2a. This distinction in views may be applied not only on part level but even on top level (“whole product” level). For example, there may be region or even customer specific versions of a product. But again they are all based on the same functional design (see Figure 2b).

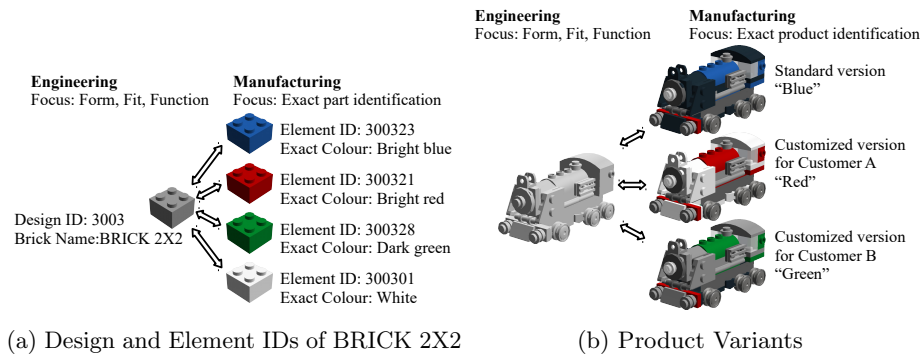


Fig. 2: Engineering vs. Manufacturing View

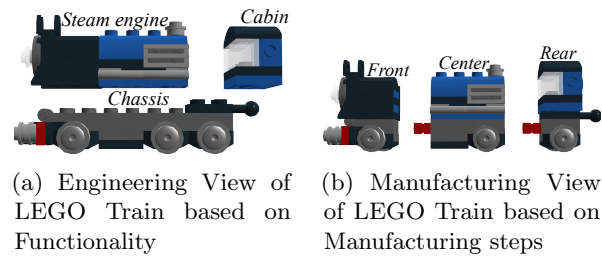


Fig. 3: Different Structure Views on LEGO Train

## 2.2 Engineering and Manufacturing Product Structure

Especially for complex products, such as in automotive, aviation, and similar sectors, the BOMs in engineering and manufacturing context differ significantly. In engineering, the product structure is deduced from the functionality of a product. The example train model could be divided into a steam engine, chassis, and operator's cabin (Figure 3a). In a real life setting, the engineering product structure might be divided into hydraulics, electrics, power train, body parts, etc. The manufacturing structure on the other hand is compounded by subassemblies, as they exist in assembling processes. For the example model, the official building instructions define the three segments front, center, and rear, which are first assembled separately and then joined together (Figure 3b). Although these are not the proper steps to assemble a real train, the analogy is still valid, as engineering and manufacturing product structure also differ in a real life setting. For example, a set of parts grouped by their functions, such as "all electrical elements", is useful in an EBOM, but will not be used in an MBOM as this group will never be assembled into a physical sub-assembly.

### 3 Requirements

Based on the scenario described above, we have derived the functional and non-functional requirements for our core ontology. When designing models for information systems, Part-whole relations, i. e., the relations between a “whole” and its parts, components and/or constituents, are of major importance [11]. Section 2.2 illustrated such varying product structures. Semantically, it would be slightly more precise to use the term “subcomponent” instead of “subpart”, as “is component of” usually does not imply transitivity, while “is part of” often does [12]. However, the latter is the commonly used term in PLM and is therefore adapted in this paper.

**Req 1: Differentiating between product concepts and product instances** Typically, the term “product” is used both as the concept of a product, like a model, drawing or idea, as well as its physical implementation, such as a concrete train with a certain ID where parts like a concrete steam engine with a specific ID are built in. In this regard, the ontology must be able to explicitly differentiate between such product concepts and product instances.

**Req 2: Different views on parts depending on context** Views on the parts used in products differ depending on PLM-phases and parties. A supplier may consider a part as an assembly, while the manufacturer considers it a subpart without a sub-structure relevant to the manufacturer. Different requirements exist regarding the information and models needed and created for the parts as these depend heavily on the phases. For example, in the early design phase, parts may not need a parameter set for their procurement method, while this information is required for production planning in the early manufacturing phase. The ontology must support separate views on the product data for the several PLM phases.

**Req 3: Distributed workflow models and workflow executions** In PLM, information about the product and the product itself change across the phases. An ontology needs to support workflows regulating the processes inside a business and across parties. Information may be accessed, validated, changed, transmitted, or taken as input for manufacturing steps, e. g., assembling a sub-assembly according to an MBOM. Workflows supporting these processes need to be weakly structured in some cases, e. g., for dynamic processes in the early development of a new product, while they need to be strongly structured in others, e. g., approval steps for changing manufacturing documents of a product already in production or even use. The ontology must therefore support workflows for using, creating, and changing product information.

**Req 4: Secure distributed group management and access right management** PLM includes providing the correct information to the authorized person at the right time. Therefore, another key-aspect of modern PLM is ensuring the correct access rights to protect the confidentiality, integrity, and availability [13] of all necessary information. This includes a flexible group management that provokes and revokes access rights to specific information, such as drawings or models, on product parts. Distributed settings including international settings have to be considered, as the parties often reside in different countries where

different legislations are applied. The ontology must support establishing rulesets for rights management.

**Non-functional Requirements:** The central purpose of a core ontology is to provide a formal foundation for data integration in a heterogeneous, possibly distributed infrastructure [14] such as found in PLM applications. Thus, besides the above mentioned functional requirements, also a couple of non-functional requirements need to be supported by CO-PLM. These non-functional requirements are derived from the experiences in designing core ontologies and are precise semantics, modularity, extensibility, reuseability, and separation of concerns [14]. CO-PLM needs to provide precise semantics in order to ensure interoperability with other ontologies. While the CO-PLM ontology can be used as is, there may be certain circumstances where it is desirable to reuse or extend only parts of it. Modularity is a pre-requisite for extensibility, i. e., the addition of future functionalities or the integration of new domain-specific knowledge. The requirement of extensibility is needed for CO-PLM in order to apply it in different sectors such as automotive, aviation, construction, etc. Finally, with separation of concerns we refer to the requirement that the CO-PLM ontology shall be applicable in arbitrary application domains that deal with production [14]. This supports the adoption of the ontology.

## 4 Solution: CO-PLM Core Ontology

Our solution to the requirements stated above is CO-PLM, a novel core ontology for PLM. Unlike existing PLM approaches, our CO-PLM extends the state of the art by supporting all phases of a product's lifecycle.

**Design Approach:** The design of CO-PLM follows the most common approach in the PLM literature by applying a part-centred model since product parts are the objects inherently holding most of the product data [15]. A part is a specific entity in the product, not the documents related to the product. In general, data for the different lifecycle phases is stored in separate software systems being updated by different departments. These systems expand in functionality over time and focus on the individual requirements of the particular department. Therefore, product models, data formats, and supported processes often differ among these systems. However, a manufacturing company must ensure all product data to be precisely assignable to a product or its parts and be shared across the different departments. CO-PLM offers a semantic basis to formalize product information and the relations among them. To this end, we base the design of CO-PLM on the foundational ontology DOLCE+DnS Ultralite (DUL) [16], because it provides rich semantics. Furthermore, we apply a pattern-based ontology design because it supports modularity by enabling reusing individual aspects of the ontology separately. The following presents an overview of its patterns.

**Fulfillment of Functional Requirements:** CO-PLM consists of ten ontology design patterns, which are implemented in six OWL files (see Figure 4) presented in the next sections. The legend depicted in this figure also applies

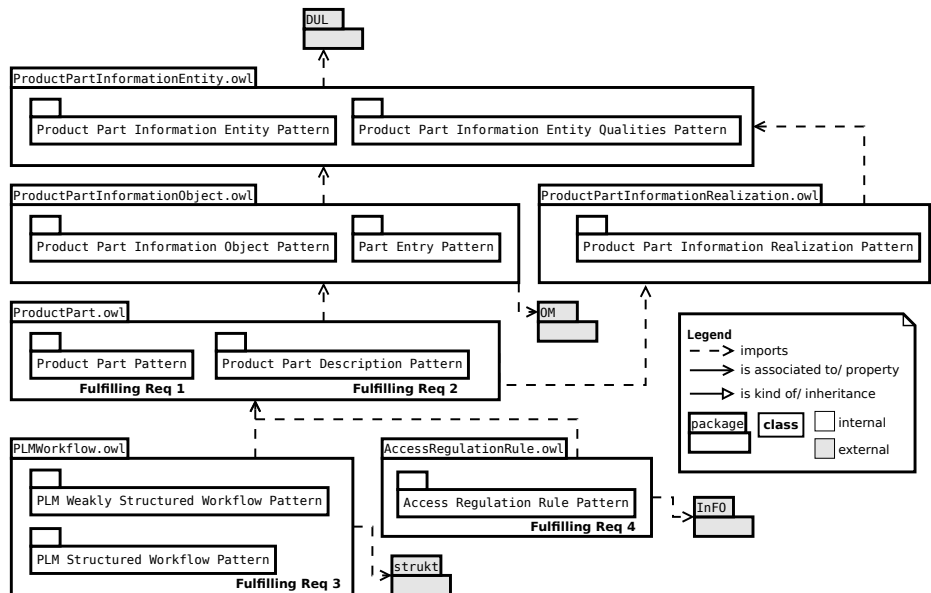


Fig. 4: Pattern Overview

for the following figures. First, the Product Part Information Entity Pattern and the Product Part Information Quality Pattern refine the Information Entity and Quality concepts from DUL in the PLM context. Next, the Product Part Information Object Pattern and the Part Entry Pattern further specify these concepts to elaborate on BOMs and their relation to referenced parts. Finally, the Product Part Pattern and the Product Part Description Pattern utilize the previous concepts as well as DUL’s Description and Situations Pattern to offer a framework for representing product information. As Figure 4 shows, the patterns are connected by imports and thereby reuse formerly defined classes. For example, `ProductPartInformationObject` defined in the Product Part Information Entity Pattern is reused in the Product Part Information Object Pattern.

The Product Part Pattern fulfils **Req 1** and the Product Part Description Pattern fulfils **Req 2**. **Req 3** is fulfilled by integrating the workflow core ontology `strukt` [17], which is also based on DUL. **Req 4** is fulfilled by a combination of `InFO` [18] for access control and `dgFOAF` [19] for group management. Thus, CO-PLM covers all of the presented functional requirements. The following sections focus primarily on the first two functional requirements, as these relate to the most essential concepts of the ontology. The **Technical Report for this paper**, describing all CO-PLM patterns in more detail, as well as OWL implementations of all patterns can be found online, at the CO-PLM repository.<sup>4</sup>

**Product Part Information Entity Pattern** In DUL, an information object represents a piece of information in its abstract form while an information

<sup>4</sup><https://github.com/FSCHOEN/CO-PLM>

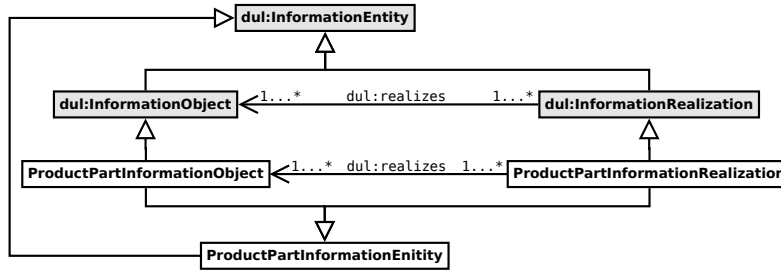


Fig. 5: Product Part Information Entity Pattern

realization is its physical manifestation, such as a paper document or a digital file, which consists of physical bits on a storage or transport medium. The central concepts of this pattern are `InformationEntity` and its subclasses `InformationObject` and `InformationRealization`. Based on these concepts, we introduce product-part-related counterparts, i. e., `ProductPartInformationEntity`, `-Object`, and `-Realization` (see Figure 5). Similar to information entities, product part information objects can have one or multiple product part information realizations and vice versa, i. e., one realization can realize one or multiple information objects.

**Product Part Information Entity Qualities Pattern** `ProductPartInformationEntities` have special `Qualities` relevant in the PLM context (see Figure 6). In DUL, a `Quality` is “any aspect of an Entity (but not a part of it), which cannot exist without that Entity” (see documentation at [16] in DUL.owl). Qualities of a `ProductPartInformationEntity` are the approval status, the level of confidentiality, and a version. The approval status illustrates the state of development such as “draft”, “approved for procurement”, “approved for prototyping”, “approved for production”, or “deprecated”. Possible values of the level of confidentiality are “public” or “company confidential” as well as governmental classification levels. A `Quality` generally belongs to exactly one `ProductPartInformationEntity`, as otherwise a change of a quality of one entity implies a change in the quality of another. In special cases this could be desirable, e. g., if multiple entities can only be approved together. `ProductPartInformationEntity` has in most cases only one `Version`, `ApprovalStatus` and `LevelOfConfidentiality`. However, there may be exceptions, e. g., if multiple systems of confidentiality levels are used in parallel. This could be an internal company standard next to governmental levels, which do not necessarily correlate. Similarly, the version could be a composition of separate versions for different countries and yet multiple versions for each country. The approval statuses can have different values depending on the state of progress or approval by different boards.



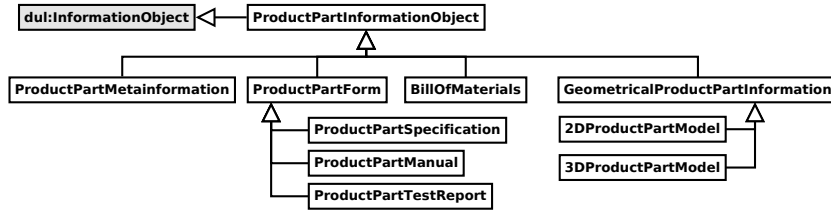


Fig. 7: Part Information Object Pattern

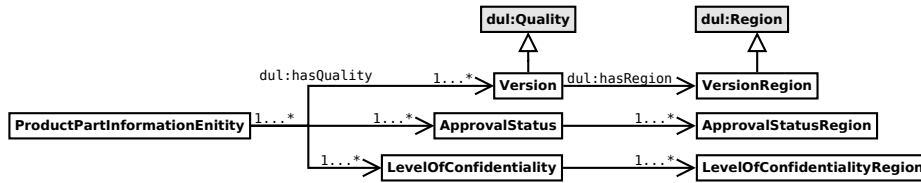


Fig. 6: Product Part Information Entity Qualities Pattern

**Product Part Information Object Pattern** The Product Part Information Object Pattern (Figure 7) illustrates the relevant information objects of the CO-PLM ontology. `ProductPartInformationObjects`, as defined in the Product Part Information Entity Pattern, are all information objects related to a part (in PLM literature often called “product data” [1,9]). The most common additional `ProductPartInformationObjects` are `ProductPartMetainformation`, such as identifiers, like the `PartNumber`, `Weight` and `Dimensions`. The `BillOfMaterials` describes the phase-dependant structure of `ProductPartInformationObject` and states its parts. `GeometricProductPartInformation` contains spatial layouts of the part and its components. `ProductPartForms` subsume information commonly represented by forms, e.g., `ProductPartSpecification`, `ProductPartManual`, and `ProductPartTestReport`.

**Part Entry Pattern** Figure 8 introduces the class `PartEntry` to represent individual entries in BOMs, using the “Ontology of units of Measure” (OM)<sup>5</sup> to represent quantities. A `Measure` is a combination of a value and a unit, e.g., “10 kg”, “5.2 m” or “3 pieces”. Also, a part entry can refer to the more general product part information entity, as e.g., a manual could be content for a sales bill of materials, describing the deliverables. The `hasConstituent` property is an oversimplification of the relationship between the product part and the BOM (or any information object describing it). Figure 11 elaborates this further.

**Product Part Pattern** A product part can either be a product part master or a physical product part (Figure 9). While a product part master describes product parts in an abstract way, physical product parts represent the real

<sup>5</sup><http://www.ontology-of-units-of-measure.org/vocabularies/om-2/> Ontologies for units are discussed at [20].

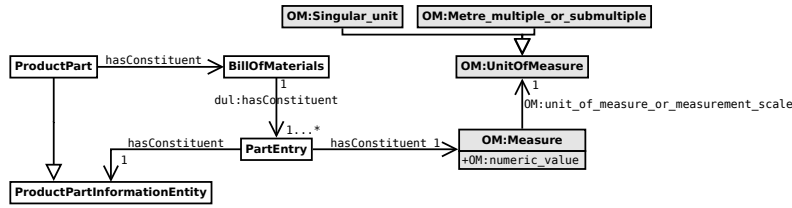


Fig. 8: Part Entry Pattern

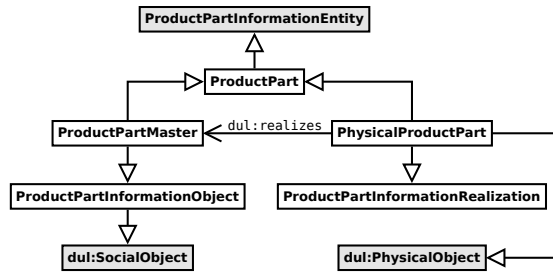


Fig. 9: Product Part Pattern

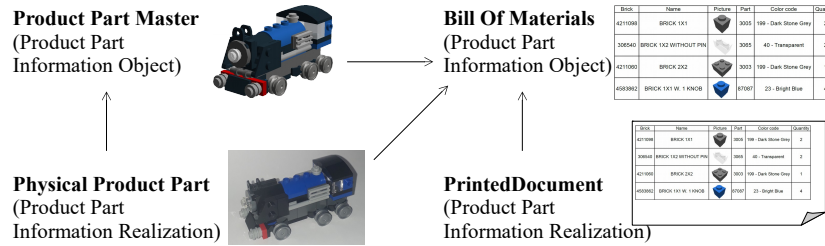


Fig. 10: Train Example Product Part Information Entities

physical objects. This pattern therefore fulfills **Req 1**. The physical objects are the realization of the master, which only exists for communication purposes. Using the example model from Section 2, Figure 10 shows different product part information entities. While the product part master in the upper left represents the LEGO model 31504 in general, the photo on the bottom left depicts one specific physical build of this model. On the right side a BOM is on the top, while its physical representation is on the bottom in form of a printed document.

Both masters and physical product parts can have subparts, which are listed in their respective BOMs. While the master can only have other masters in its BOM, a physical product part can have both masters and physical product parts in its BOM. The BOM of a master, which is the basis for multiple physical product parts, cannot contain a specific physical subpart. On the other hand, the BOM of a physical product part can either refer to a subpart master (anonymous



## 5 Evaluation of Practical Use

The fulfillment of the functional and non-functional requirements was shown in Section 4. This section evaluates the practical use of CO-PLM in the context of an application in a vehicle manufacturing company. Using ontologies in practice requires an integrated engineering process, which is closely aligned to system development. Therefore, we first describe our approach of combining ontologies and software design, before we further discuss technical evaluation of CO-PLM.

**Combined Engineering Process:** CO-PLM is embedded in an advanced joined ontology engineering process and a software engineering process. As ontologies are used to create and query triples from software applications, the ontology engineering process has to be aligned to the software engineering process. This ensures maintainability, reusability, and extensibility of the developed software artifacts and ontologies [21]. Although ontologies and software are used in combination with each other, their development is slightly different. Ontologies are data models and follow a data-driven modeling approach. In contrast, software implements behavior and focuses on functionality. For example, ontology design patterns are only relevant within an ontology and not all of their details must be mapped to source code. Instead, a software's source code should provide an easy to use programming interface to create the individual triples of an ontology design pattern. On the other hand, functions like `get`, `set`, `update`, and `execute` are required in a software implementation, but are not part of an ontology. Approaches exist to combine ontology engineering and software engineering, such as TwoUse [21,22]. TwoUse is an open source tool implementing OMG and W3C standards in order to develop ontology-based software models as well as model-based ontologies. However, additional research has to be done to better integrate such approaches in modern system engineering processes including test, configuration, build, change, and artifact management, which are not considered in current approaches.

Our solution for such an integrated engineering process is depicted in Figure 12. First, ontologies are created as OWL files using an ontology editor such as Protégé<sup>6</sup>. In order to combine ontologies with software artifacts, we use Apache Maven<sup>7</sup> for managing both our ontology and software projects. Maven uses a project object model (POM) to store all information of a particular project including the project's name and its connections to other projects. Second, OWL files and POM files are stored in a version control system, which also stores the software's source code. Third, any changes to the version control system triggers a build server to download the latest version of the ontology and to perform a build process. The build process generates an API which can be used by software applications to create triples. Finally, all files of the ontology project are stored in an artifact repository which is used by both ontology engineers and software developers to retrieve any required artifact. In order to use an ontology, a software developer

---

<sup>6</sup><https://protege.stanford.edu/>

<sup>7</sup><http://maven.apache.org/>

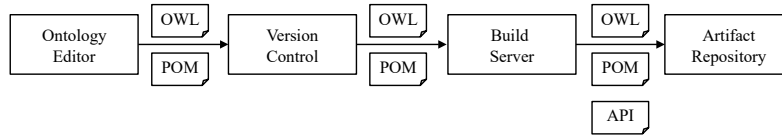


Fig. 12: Semantic Engineering Process combining Software and Ontology Engineering

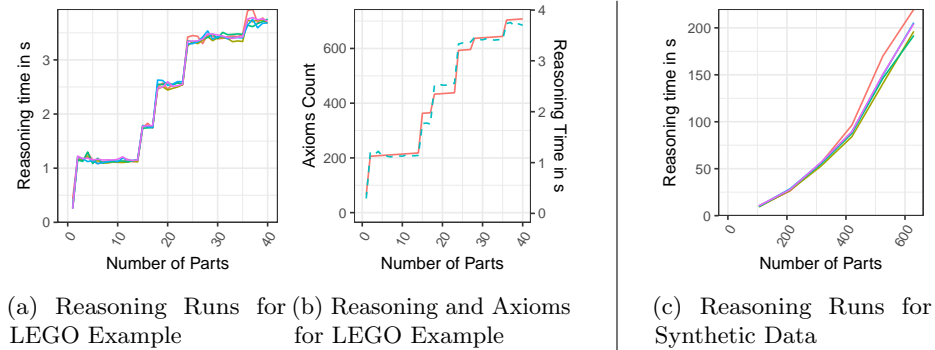


Fig. 13: Reasoning Time related to Number of Part Types

imports the ontology’s API in the source code. This is done by defining a Maven dependency to the API using the POM file of the software artifact.

**Evaluation on Demo LEGO Train Example Data:** To evaluate the practical use of CO-PLM, the example model from Section 2 was implemented and checked for consistency with the HerMiT OWL Reasoner [23]. By implementing the model step-wise, one part at a time, we can observe how reasoning time increases with growing part counts. Figure 13a shows five of these reasoning runs. The plot shows intervals of almost no gain in reasoning time as well as sudden jumps. The plateaus appear when adding primitive parts without subparts, while the jumps originate from adding subassemblies. When plotting the mean time (dotted line in Figure 13b) of these runs together with the axiom count, a significant correlation of these two graphs becomes apparent (Figure 13b). A linear regression model fits with  $R^2 = 0.99$ ,  $p < .00001$  and  $df = 38$ . This suggests a linear correlation between the reasoning time and the amount of parts, as the latter directly influences the amount of axioms.

**Evaluation on Large-Scale Synthetic Data:** A passenger car usually has around 30,000 individual parts,<sup>8</sup> while an airline plane even has ca. six million individual parts.<sup>9</sup> As such real-life product data structures are not publicly available, we use synthetic data to simulate a real-life product. An important

<sup>8</sup><http://www.toyota.co.jp/en/kids/faq/d/01/04/>, last accessed: 07/16/2018

<sup>9</sup><http://magazin.lufthansa.com/xx/en/fleet/boeing-747-8-en/one-plane-six-million-parts/>, last accessed: 07/16/2018

insight is that the linear correlation appears to not hold anymore with larger part counts, as Figure 13c illustrates. It depicts five reasoning runs on automatically generated product structures. Here, the experiments start with 100 part types and 5 subassemblies. Each of the subassemblies has 20 different subpart types (covering all of the 100 part types). Each subpart type is used 3 times in a subassembly. The first point on the graph in Figure 13c represents a product with 300 individual part pieces, 100 different part types (à 3 pieces each) and 5 subassemblies. Then, the amounts of parts and subassemblies are increased by the same values at each step. Thus, the second experiment is run on 600 part pieces, 200 part types and 10 subassemblies, the third on 900 part pieces 300 part types and 15 subassemblies, etc. Three pieces per subpart are not relevant for the reasoning but illustrate that 100 part IDs correspond to 300 individual parts if every part ID is used 3 times on average in the product. Each “chunk” of 100 part types including 5 subassemblies can be thought of as an independent package, e. g., an assembled engine or a seat, of instantiated concepts of CO-PLM.

A quadratic regression model ( $R^2 = 0.9973$ ,  $p < 0.001$  and  $df = 3$ ) fitted to the mean of the runs in Figure 13c predicts the reasoning time for a whole car to take around 16 hours. For the airplane the estimation is 71 years. It becomes obvious that reasoning on whole products is not practical. In this example, every 100-part-chunk has the same structure. Therefore, in an optimal process, the independent reasoning of, e.g., 1.000 parts (10 chunks) should take 10 times the time of 100 parts (1 chunk). In such a case, reasoning the car triples would only take 17 minutes and 7 days for the plane. There, an approach for improving the reasoning time is needed. By enabling reasoners to recognize instances of ontology patterns and then process these separately, reasoning could be accelerated. As shown above, reasoning 10 chunks/ instances of ontology patterns separately is much faster than reasoning the same amount of data in one run.

## 6 Conclusion

This work presented the core ontology CO-PLM to formalize product part information across all lifecycle phases from idea generation to disposal. CO-PLM is based on the foundational ontology DOLCE+DnS Ultralite. We evaluated the ontology in regard to its fulfillment of the functional and non-functional requirements and analyzed the reasoning times in practical use with increasingly complex products. For future work, CO-PLM will be extended by an advanced distributed group management and secure access control in multi-national industrial projects.

## References

1. J. Stark, *Product lifecycle management: 21st century paradigm for product realisation*. Decision engineering, London and New York: Springer, 2nd ed. ed., 2011.
2. X. G. Ming, J. Q. Yan, X. H. Wang, S. N. Li, W. F. Lu, Q. J. Peng, and Y. S. Ma, “Collaborative process planning and manufacturing in product lifecycle management,” *Computers in Industry*, vol. 59, no. 2-3, pp. 154–166, 2008.

3. S. Foufou, S. J. Fenves, C. E. Bock, S. Rachuri, and R. D. Sriram, *A Core Product Model for PLM with an Illustrative XML Implementation*. NIST, 2005.
4. H. Panetto, M. Dassisti, and A. Tursi, "Onto-pdm: Product-driven ontology for product data management interoperability within manufacturing process environment," *Advanced Engineering Informatics*, vol. 26, no. 2, pp. 334–348, 2012.
5. M. Imran and R. Young, "Reference ontologies for interoperability across multiple assembly systems," *IJPR*, vol. 54, no. 18, pp. 5381–5403, 2015.
6. P. Leitão and F. Restivo, "Adacor: A holonic architecture for agile and adaptive manufacturing control," *Computers in Industry*, vol. 57, no. 2, pp. 121–130, 2006.
7. G. Bruno, D. Antonelli, and A. Villa, "A reference ontology to support product lifecycle management," *Procedia CIRP*, vol. 33, pp. 41–46, 2015.
8. A. Matsokis and D. Kiritsis, "An ontology-based approach for product lifecycle management," *Computers in Industry*, vol. 61, no. 8, pp. 787–797, 2010.
9. S. G. Lee, Y.-S. Ma, G. L. Thimm, and J. Verstraeten, "Product lifecycle management in aviation maintenance, repair and overhaul," *Computers in Industry*, vol. 59, no. 2-3, pp. 296–303, 2008.
10. R. D. Reid and N. R. Sanders, *Operations management: An integrated approach*. Hoboken, NJ: John Wiley & Sons Inc, 4th ed. ed., 2010.
11. N. Guarino, S. Pribbenow, and L. Vieu, "Modeling parts and wholes," *Data Knowl. Eng.*, no. 20(3), pp. 257–258, 1996.
12. A. C. Varzi, "Parts, wholes, and part-whole relations: The prospects of mereotopolog," *Data Knowl. Eng.*, no. 20(3), pp. 259–286, 1996.
13. J. Andress, "What is information security?," in *The Basics of Information Security*, pp. 1–16, Elsevier, 2011.
14. A. Scherp, C. Saathoff, T. Franz, and S. Staab, *Designing core ontologies*. IOS Press, Applied Ontology 6 (2011) 177–221, 2011.
15. W. Liu, Y. Zeng, M. Maletz, and D. Brisson, *Product Lifecycle Management: A Survey*. ASME 2009, 2009.
16. A. Gangemi, *DOLCE+DnS Ultralite (DUL)*. ontologydesignpatterns.org, 2016.
17. A. Scherp, D. Eißing, and S. Staab, *strukt - A Pattern System for Integrating Individual and Organizational Knowledge Work*. ISWC 2011, pp. 569-584. Springer, Berlin, Heidelberg, 2011.
18. A. Kasten and A. Scherp, "Ontology-based information flow control of network-level internet communication," *IJSC*, vol. 9, no. 01, pp. 1–45, 2015.
19. F. Schwagereit, A. Scherp, and S. Staab, *Representing Distributed Groups with dgFOAF*. Heraklion, Crete, Greece: ESWC 2010, 2010.
20. M. D. Steinberg, S. Schindler, and J. M. Keil, "Use cases and suitability metrics for unit ontologies," in *OWL: experiences and directions - reasoner evaluation* (M. Dragoni, M. Poveda-Villalón, and E. Jimenez-Ruiz, eds.), vol. 10161 of *LNCS*, Springer, 2017.
21. F. S. Parreiras, *Marrying model-driven engineering and ontology technologies: The TwoUse approach*. PhD thesis, University of Koblenz-Landau. URL: <http://kola.opus.hbz-nrw.de/volltexte/2011/599/pdf/book.pdf>, last accessed: 07/16/2018, 2011.
22. T. Walter, *Bridging Technological Spaces: Towards the Combination of Model-Driven Engineering and Ontology Technologies*. PhD thesis, University of Koblenz-Landau. University of Koblenz-Landau. URL: <http://kola.opus.hbz-nrw.de/volltexte/2011/682/pdf/book.pdf>, last accessed: 07/16/2018, 2011.
23. R. Shearer, B. Motik, and I. Horrocks, *HermiT: A Highly-Efficient OWL Reasoner Directions*. ISWC 2008. Springer, Berlin, Heidelberg, 2008.