INTERNATIONAL DESIGN CONFERENCE - DESIGN 2018
Dubrovnik - Croatia, May 21-24, 2018.

ONTOLOGY-BASED APPROACH FOR THE USE OF INTENTIONAL FORGETTING IN PRODUCT DEVELOPMENT

[Authors will be inserted automatically]

Abstract

[Abstract will be inserted automatically]
[---]
[---]
[---]
[---]
[---]
[---]
[---]
[---]
[---]
[---]
[Do not delete or modify the abstract area]

[Keywords will be inserted automatically]
[---]

1. Introduction

Forgetting is usually considered to be negative. To forget something seems not to be helpful, especially in light of the fact that knowledge seems to become increasingly valuable as a company resource. However, our memory might be overwhelmed, if we could not forget. According to Douwe Draaisma, the main reason for this is not that the memory capacity is too low, like a memory card, but because we would not be able to distinguish between important and unimportant facts (Kara, 2015). Moreover, single experiences form our empirical knowledge, even though the certain situations of these experiences are forgotten (Kara, 2015). The knowledge, we are interested in here, is called implicit, which means the knowledge is linked to a person, foremost non-verbal and non-formalizable, e. g. in speech and writing (VDI 5610, 2009). According to this, these mechanisms of forgetting might also be called implicit, because the forgetting happens unconsciously and intuitively, for instance, when the implicit knowledge is not relevant anymore or replaced with new experiences. Thus, the question to be asked is, how could these implicit forgetting mechanisms be made explicit and thus intentionally usable? To answer the question, it first has to be clarified what explicit means with regard to knowledge. In VDI 5610 (2009) knowledge is defined as explicit, if it is formalizable in language or script and so it can be communicated. In companies, it is usually hold in data and knowledge bases. So, the question might even be whether and how to get those data- and knowledgebases to forget explicit knowledge intentionally. But, if knowledge is an important resource for companies, why should it be forgotten?

The reuse of product and process models during a product development process leads to considerable saving of cost and time (Albers, 2015). As a consequence, the company databases grow bigger while storing the documentation of closed development projects. Suppose there is a new product development task and particular knowledge of these previously completed projects might be reused, the developers first have to filter the relevant elements from the large amount of contents and reject the irrelevant pieces. However, most data and knowledge bases are designed to provide comprehensive knowledge, thus the developers use the described implicit forgetting mechanisms unconsciously for filtering the contents. This analogously leads to the problem of overwhelmed memory as described afore. The knowledge reuse might be more efficient, if the naturally and implicitly used forgetting mechanisms can be applied in an intended way and are supported within methods and applications for product development. This supports developers in distinguishing the relevant and irrelevant elements of product knowledge for the current development task. In the following, the method of supportive removal in knowledge bases is called *Intentional Forgetting*, because it is inspired by the described implicit and unconsciously used forgetting mechanisms.
This contribution introduces a novel approach for supporting product development processes with operators for Intentional Forgetting (IF). For this, product models and descriptions have to be automatically mapped to ontologies, thus the supportive IF-mechanisms can be implemented by using the SPARQL update query language. The approach supports designers in the systematic reuse of existing design knowledge and thus opens new possibilities for knowledge-based engineering. The support by removing obsolete elements for reuse is demonstrated in a use case of a test-rig design process, which is characterized by a high level of knowledge reuse from an earlier test-rig.

2. State of the art and background

In the following, knowledge representations are introduced. Especially, the basic idea of the ontology-based knowledge representation and its fundamental definitions combined with languages enabling to query and update the represented knowledge is shown, which are used to implement the IF-operators.

2.1. Knowledge representation in product development

The most common approaches to represent knowledge in product development are rule-based, object-oriented and constraint-based methods. While rules merely consist of an "if-then-else"-structure, which connects facts in a procedural way, object-oriented systems structure objects in domains (Rude, 1998). For technical issues, constraints are often used, because they can represent restrictions or relations between objects (Rude, 1998). For more complex circumstances with many different relations, these simple representations are often not sufficient because dependencies cannot be mapped appropriately. Therefore, semantic networks, taxonomies and especially ontologies are more appropriate. A taxonomy structures terms hierarchically in a class-subclass-system, whereas semantic networks have the highest semantic integration of relations between objects (Kienreich and Strohmaier, 2006). An ontology possesses all these qualities and constitutes a powerful way to represent general knowledge. A well-known formal definition comes from Gruber (1993), who defines an ontology as an explicit specification of a (shared) conceptualization. *Explicit* is related to the explicitly defined concepts and relations. *Shared conceptualization* refers to the fact that the represented knowledge is agreed upon by the respective group. In practice, ontologies are usually formalized in the Web Ontology Language (OWL), which is based on description logics (Baader, 2017) and developed by the World Wide Web Consortium (W3C), an international community that standardises web technologies. Knowledge represented in description logics is usually divided into a TBox and an ABox. The TBox contains the terminological knowledge and consists of axioms. These axioms represent knowledge in the form of classes and binary relations that connect these classes (e. g. Figure 2 in Section 3.2). The ABox contains assertions about individuals and describes to which classes these individuals belong to and how they are connected via relations. A TBox axiom can be written as "**gear_unit** ***hasPart*** *some* **shaft**", where **gear_unit** and **shaft** are the classes and ***hasPart*** is a relation between them. An ABox axiom is "**08.100** ***hasPart*** **08.100.001.001**", where **08.100** is an assembly name and denotes an individual of the type **gear_unit** and **08.100.001.001** is a component name and is defined as an individual of the class **shaft**. The identification numbers are used to identify the certain components. Ontologies represent not only explicitly stated knowledge, but also knowledge which can be inferred from the stated knowledge. Thus, ontological knowledge bases have a larger content than only the stated input. This inferred knowledge is also taken into account in the IF-operators (Section 3.3).

2.2. Query OWL ontologies

Once knowledge is represented, it is important to have the possibility to access the elements of the ontological knowledge base. The SPARQL-1.1 language (Seaborne and Harris, 2013) was developed to meet these requirements by providing the possibility to query OWL ontologies. The following SPARQL query can be used to determine everything which is a part of a gear unit.

```
SELECT  ?Y
WHERE  { ?X  a :gear_unit.
         ?X   :hasPart   ?Y.}
```

In this query *?X* and *?Y* are variables, which are bound to individuals occurring in the ontology. This binding is carried out in a way such that the condition specified in the *WHERE* clause is fulfilled. More specifically, *?X* is bound to some individual i_1 belonging to class **gear_unit** and *?Y* is bound to some individual i_2, which is connected to i_1 via the ***hasPart*** relation. The *WHERE* clause of the query determines all possible candidates for i_1 and i_2. The *SELECT* clause then presents all possible bindings for *?Y* as result. Besides accessing the information in the ontology, another challenge in the daily use of ontologies is dynamic knowledge. Usually, the represented knowledge is not static but subject to frequent changes and even the development of an ontology can be seen as an evolutionary process. For example, in product development, the documentation of existing products is often reused and adapted to new products. This illustrates the need for dynamics in knowledge bases in the area of product development. To meet the requirements for dynamics, the SPARQL update language (Gearon et al., 2013) was developed. With the help of SPARQL update queries, outdated knowledge can be deleted and new contents can be inserted into an ontology. The following query can be used to replace all feather keys which are part of a module by splined shafts:

```
DELETE { ?Y  a :feather_key   }
INSERT  { ?Y  a :splined_shaft }
WHERE  { ?X  a :module.
         ?X   :hasPart   ?Y.
         ?Y  a :feather_key   }
```

In general, there might be several ways to specify the result of such a SPARQL update query. With the help of different semantics for SPARQL update, it can be specified which of these possible outcomes represents the desired result for the application at hand. This paper sticks to the semantics introduced by Schon and Staab (2017) and furthermore introduces some new semantics suitable for the application in product development.

3. Intentional Forgetting in ontology-based product representations

In this section, an approach for an ontology-based product representation is introduced with focus on CAD data. Initially, the workflow and methods applied are presented. Then an automated approach for extraction of CAD representations into an ontology-based representation in Protégé (https://protege.stanford.edu/) is shown, which enables the subsequently presented IF-mechanisms. Moreover, the IF-mechanisms, which are of interest for product development, are introduced and exemplarily implemented by using SPARQL update queries.

3.1. Workflow and methods applied

For the assistance by IF-mechanisms in product development, essential IF-operators are conceptualized and implemented in an ontological knowledge base system. These operators are based on the analyses of forgetting mechanisms, which are already implicitly used by developers to identify relevant elements from previous development processes. These unconsciously and mentally used mechanisms are called IF-mechanisms, as described afore. However, IF-mechanisms are not yet supported by procedure models or IT systems. With the focus on automotive and test-rig engineering, the example in Figure 1 shows the product development process of a gear unit according to Pahl and Beitz (1996). As shown in Figure 1, not only CAD models are considered in these analyses. All development phases and their resulting product descriptions (in any degree of concretization) are taken into account: starting with the requirements (product planning) and derived functional structures and principal solution (conceptual design) up to the resulting CAD models (embodiment design) and manufacturing documents (detail design). Thus, the entire development process can be supported by IF. As shown in Figure 1, these informal product and process descriptions have to be transferred into formal ontology-based models to provide a consistent linkage between the single work results of the process. To efficiently develop these ontologies, procedures have to be established to automatically generate them from available databases in product development, for instance from CAD models. Finally, a dynamic ontological knowledge base is set up and IF-operators are integrated in the system, which support developers with an efficient reuse of product and process models by the removal of obsolete elements. (Kestel et al., 2017)

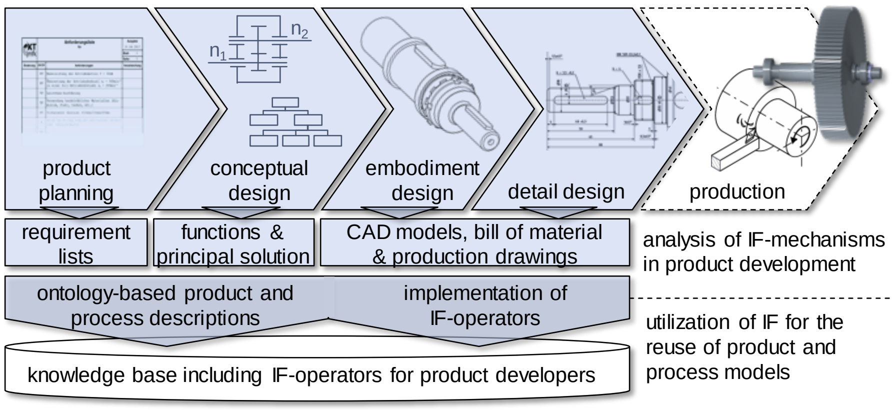

product planning
conceptual design
embodiment design
detail design
production
requirement lists
functions & principal solution
CAD models, bill of material & production drawings
analysis of IF-mechanisms in product development
ontology-based product and process descriptions
implementation of IF-operators
utilization of IF for the reuse of product and process models
knowledge base including IF-operators for product developers

Figure 1. Supporting reuse by IF-operators according to (Kestel et al., 2017)

3.2. Ontology-based approach for product representations

In the following, the focus is on the embodiment design phase. Classes, relations and instances, which were introduced in Section 2.1, are shown exemplarily below for a shaft from a gear unit (Figure 2). In the ontology, the classes represent a structure, which is similar to the model tree in a CAD-environment. Thus, there are the classes **assembly**, **part** and also **standard_part**, like a feather key. Other parts, like shafts, can be mapped as subclasses of **part**, thus a specific shaft, in this case the shaft **08.100.001.001**, can be implemented as an individual of the type **shaft** (Figure 2). Binary relations are set between the classes, e. g. ***hasPart*** to express that an assembly has some parts. Moreover, relations can be set in both direction, like ***isPartOf*** in Figure 2, which is the inverse of ***hasPart*** and can analogously be set for the other ***hasPart***-relation (For reasons of clarity, however, this has been dispensed with). Sometimes, it can be helpful to include a class for **component_geometry**, for instance the special keyway associated with the feather key can be related to the shaft.

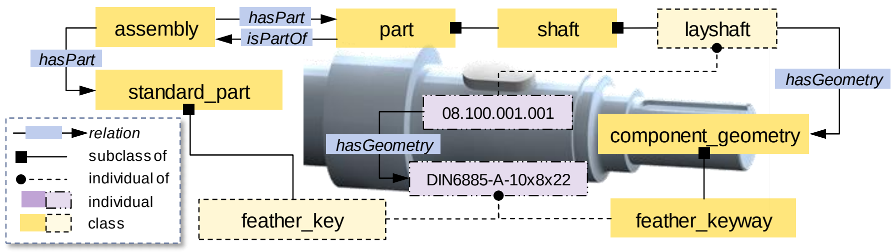
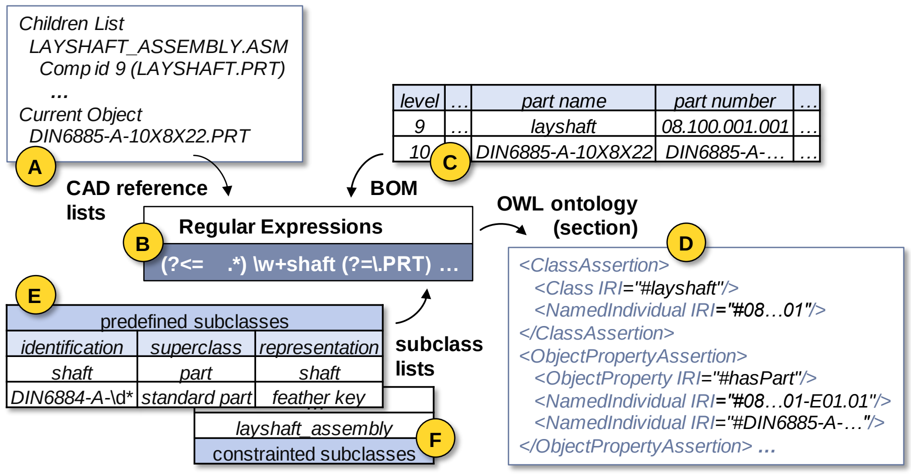


Figure 2. Example for an ontology-based product representation with pre-defined and generated (dashed boxes) classes, individuals and relations

A procedure to automatically derive ontologies from a CAD model uses reference lists and bills of material (BOM) exported from the CAD system, thus the manually effort is reduced. This procedure is implemented in the environment of MathWorks MATLAB and PTC Creo Parametric. An excerpt of an exported CAD reference list is shown in Figure 3 A. These reference lists are incrementally generated for every component in a CAD model. Each list includes the currently considered object (e. g. a part) as well as all assemblies and parts attached to this object. Indentations in the reference lists represent assembly hierarchies. In the given example, an assembly and its layshaft component (see *"Children List"* in Figure 3 A) are attached to a feather key (*DIN6775-A-10X8X22.PRT*).



Children List
LAYSHAFT_ASSEMBLY.ASM
Comp id 9 (LAYSHAFT.PRT)
...
Current Object
DIN6885-A-10X8X22.PRT
A
CAD reference lists
level | ... | part name | part number | ...
9 | ... | layshaft | 08.100.001.001 | ...
10 | ... | DIN6885-A-10X8X22 | DIN6885-A-... | ...
C
BOM
Regular Expressions
B
(?<= .*) \w+shaft (?=\.PRT) ...
OWL ontology (section)
D
<ClassAssertion>
<Class IRI="#layshaft"/>
<NamedIndividual IRI="#08…01"/>
</ClassAssertion>
<ObjectPropertyAssertion>
<ObjectProperty IRI="#hasPart"/>
<NamedIndividual IRI="#08…01-E01.01"/>
<NamedIndividual IRI="#DIN6885-A-…"/>
</ObjectPropertyAssertion> ...
E
predefined subclasses
identification | superclass | representation
shaft | part | shaft
DIN6884-A-\d* | standard part | feather key
subclass lists
layshaft_assembly
constrainted subclasses
F

Figure 3. Automated generation of ontologies

To automatically extract this product information, regular expressions are applied. Regular expressions map patterns and structures recurring in reference lists by strings and operators. These expressions can be applied to detect relevant sections in a reference list and to extract essential product information. A simplified example for such a regular expression is shown in Figure 3 B. By this expression, the layshaft component is identified in the reference list. The first part *"(?<= .*)"* searches for indentations in the reference list to identify hierarchical relationships, for instance all components assigned to the given assembly. The last part of the regular expression *"(?=\.PRT)"* uses the endings of the list entries to check whether the considered entry is a component, an assembly or a geometric feature. The name of the shaft is not completely specified in the given regular expression: the expression *"\w+"* also accepts prefixes ahead of this term, thus it is recognized, if "shaft" is contained somehow in the name. With this method a strict naming convention is not required, just the name of a part has to be used in some way. All surrounding strings identified by the bracketed expressions are not extracted, such as the ending *".PRT"* with *"(?=\.PRT)"*. The bracketed expressions are used to identify parts and their hierarchical relationships. Subsequently, strings extracted by the regular expressions are automatically transferred into OWL (Figure 3 D). These can be processed in Protégé, and can be visualized like in Figure 2 (dashed boxes). The extracted strings are used directly for the definition of subclasses. In the case of prefixes or suffixes, additional subclasses are defined. In the given example, not only **shaft** is defined as a subclass of **part**, but also **layshaft** as a subclass of **shaft** is extracted (see Figure 2). To automatically assign an individual to these subclasses, corresponding component names in the BOM are used. In the BOM in Figure 3 C, the component name of the layshaft (**08.100.001.001**) is used to create an individual for this subclass (compare ontology in Figure 2 and BOM in Figure 3 C). An example of how the resulting OWL-structures looks like is also given in Figure 3 D. Furthermore, a layshaft connection is defined in this example (Figure 3 D, "08…01-E01.01") and corresponding parts are assigned to this connection, like the feather key or attached layshaft. Since a direct definition of regular expressions for every subclass is not efficient, the implemented interface is based on a pre-defined subclass list, as shown in Figure 3 E. In the subclass lists, the first column includes terms used to identify the subclasses. These terms are inserted in the given regular expression in Figure 3 B. As mentioned before, the given terms only have to be partly included in the reference lists and prefixes or suffixes are also identified. Although **shaft** is given in the subclass list, the **layshaft** is also identified and set as subclass in the ontology (Figure 2). Regular expressions can be used in the subclass lists, for instance to identify feather keys with different dimensional parameters (and according names) by only one expression (*"DIN6884-A-\d*"* in Figure 3 E). The information extraction is based on the typical naming convention for standard parts used in standards tables. The second column of the subclass list includes corresponding superclasses and the third column specifies a representation name for a subclass in the resulting ontology. Even if no appropriate entries are found in the subclass list, components are identified by regular expressions, defined as subclasses and assigned to basic classes, e. g. to **part** or **assembly**. Thus, additional subclass lists are required to prohibit undesired classification, as shown in the constrained subclass list in Figure 3 F. Parts that are only assigned to basic classes can be further specified subsequently by knowledge engineers. Thus, the system learns the expressions of different CAD users and unites them in an ontology that is understandable to all product developers.

3.3. Intentional Forgetting with SPARQL Update

Five mechanisms are defined to efficiently support the reuse of knowledge in product development processes. The first mechanism is the *forgetting of inferred knowledge*. Within ontology-based representations, implicit knowledge can be inferred from explicitly stated knowledge. In general, there can be more than one way to infer a certain piece of knowledge from an ontology. Moreover, there can be more than one way to delete some piece of knowledge from an ontology. An example from the gear unit assembly (Figure 2) might be a TBox consisting of the single axiom "*(***part** *and* **standard_part***) SubClassOf* **component_geometry**", stating that every individual that belongs to classes **part** and **standard_part**, also belongs to class **component_geometry**. This axiom is defined, because the geometry is specified in DIN standards for standard parts and thus the geometry defines the part. Assume the ABox contains the following axioms about the individual of the feather key:

(1) **part (DIN6885-A-10x8x22)**

(2) **standard_part (DIN6885-A-10x8x22)**

(3) **component_geometry (DIN6885-A-10x8x22)**

These axioms state that individual **DIN6885-A-10x8x22** belongs to class **part** (1), **standard_part** (2) and **component_geometry** (3). Assume that axiom (3) is supposed to be forgotten, because a new geometrical element has to be defined. Deleting axiom (3) is not sufficient, since it still can be inferred from axiom (1) and axiom (2) together with the axiom in the TBox. This illustrates that deleting inferred knowledge is more challenging than just deleting a database entry (Kestel et al., 2017). More specifically, (3) has to be deleted together with all its causes. This can be accomplished in the following three ways:

deleting (3) *and* (1),

deleting (3) *and* (2) *or*

deleting (3), (1) *and* (2).

All three options prevent axiom (3) from being inferred. Different semantics for SPARQL update are used to choose one of these options. See Schon and Staab (2017) for an overview of these semantics.
In many cases, forgetting a single model element requires that connected elements in one phase or even beyond the phases have to be forgotten, too (Kestel et al., 2017). This kind of forgetting mechanism is denoted by *cascading forgetting*. In product development, it takes into account that a change of a requirement is potentially related to functions and even the parts and assemblies of a product. For instance, the requirement of the transmission of a unilateral torque has changed into an alternating torque, the feather key connection in the example of the gear unit has to be changed to a splined shaft connection. This also affects the feather key (part) and even the feather keyway (geometry), which also has to be removed. Furthermore, it could cause a change in the axial locking elements, or even the shafts diameter. Some aspects of this cascading forgetting can be accomplished by special semantics for SPARQL update. A prerequisite for implementing cascading forgetting is that certain metaproperties of the classes in the ontology are specified. Examples for these metaproperties are *rigidity* and *dependency*, which were introduced in the OntoClean methodology (Guarino and Welty, 2009). A class is called

rigid, if it is essential to all its individuals. For instance, the class person is rigid, since one cannot stop being a person. In contrast to that, the class student is not rigid, because one can start or stop being a student at any time. The definition of the set of all rigid classes of an ontology can be exploited to model a cascading forgetting. If such a deletion occurs, this indicates that all information about the individual is supposed to be removed from the ontology, resulting into the deletion of all axioms containing this individual. Another metaproperty that can be used to influence certain aspects of cascading forgetting is the dependency metaproperty. Assume that in an ontology the class **part** depends on class **assembly** via the relation ***isPartOf***, which is inverse to ***hasPart*** (Figure 2). Intuitively speaking, this means that for every individual belonging to class **part** there has to be one in class **assembly** which is connected via the ***isPartOf*** relation. Furthermore, the ABox of this ontology contains the following axioms:

(1) **part ($\mathbf{p}_1$)**

(2) **assembly ($\mathbf{a}_1$)**

(3) ***hasPart*** **($\mathbf{a}_1$, $\mathbf{p}_1$)**

Where axiom (1) denotes that individual $\mathbf{p}_1$ belongs to class **part** and axiom (3) describes that $\mathbf{a}_1$ is connected to $\mathbf{p}_1$ via relation ***hasPart***. This implies, that individual $\mathbf{p}_1$ is connected to individual $\mathbf{a}_1$ via relation ***isPartOf***. In addition, assume that there is no other individual, which is connected to $\mathbf{p}_1$ via the ***hasPart*** relation. Due to some design choices, the product developer decides to remove the assembly $\mathbf{a}_1$ (2). This removes the only individual, which is reachable from $\mathbf{p}_1$ via the ***isPartOf*** relation. Due to the open-world semantics of description logics, removing axiom (2) does not affect the class affiliation of individual $\mathbf{p}_1$. In contrast to that, the user intuitively expects axiom (1) and (3) to be deleted from the ABox as well, which corresponds to cascading forgetting. Such a cascading behaviour can be accomplished by using specific semantics for SPARQL update queries. These so-called OntoClean-guided semantics extend the query-driven semantics introduced by Schon and Staab (2017) by cascading behaviour. In addition, there are other forgetting mechanisms, which are desirable for the area of product development. A *representation of blank space* stands for obsolete elements, which are not completely removed, but rather replaced by a blank space, that facilitates the substitution with a new element (Kestel et al., 2017). In the example of a gear unit, the forgetting of all elements of the feather key connection would initially lead to the unfulfilled function "transmitting torque", because there is no connection anymore, which can comply with the requirement. In consequence, there has to be an expressive representation for this blank space, in order to ensure that the function is not deleted through IF. Furthermore, within complex product structures, it can be helpful to focus on a special part of interest. Therefore, the parts that are currently obsolete have to be hidden by *temporary forgetting*. This mechanism leads to a prioritization of the elements within certain the development steps (Kestel et al., 2017). Due to transparency and traceability, the mechanism *remembering of forgetting* takes care that additional information about the forgetting process is available like who performed for which reasons the forgetting operation at which time. This information is valuable to achieve reversibility of performed forgetting operations (Kestel et al., 2017). Also this mechanism can be exploited to document decisions and the reasons behind or moreover design alternatives in addition to the class hierarchy, for example, through annotations. Representation of blank spaces, temporary forgetting as well as mechanisms to remember forgotten aspects have not been implemented yet and are subject of current research.

4. Use Case: Development process of a cam/bucket tappet component test-rig

The introduced approaches of ontology-based knowledge representation, automated extraction of CAD data into OWL and the IF-mechanisms are now exemplarily applied to the product development process of a test-rig, which is used for studying the wear behaviour and lubrication conditions of the cam/tappet contact in combustion engines. The test-rig is suitable as a demonstrator, since its product development process is highly based on reuse and adaptation from an earlier test-rig development, thus the situations of appropriate support with IF can be analysed. In the following, the background of the test-rig is presented. Subsequently, a basic structure for an ontology-based representation in product development is shown, which can be used for the automated extraction of CAD data. Finally, the IF-mechanism *cascading forgetting* is exemplarily implemented with SPARQL update.

4.1. Experimental investigations on the friction and wear behaviour of the cam/tappet contact in the valve train

Growing demands for increasing power density and reliability of technical systems combined with dwindling resources, rising environmental awareness and stricter legislative frameworks lead to the necessity for more efficient machine and engine elements. Therefore, the reduction of friction and wear in highly stressed tribological contacts of combustion engines is vital. Besides crankshaft drive and piston assembly, the cam/tappet contact of the valve train represents one of the key systems contributing to frictional losses, especially at lower crankshaft speeds (Koch and Geiger, 1997). The valve train is responsible for the control of air and fuel flow into and out of the cylinders. The tribological system of the cam/bucket tappet contact is shown schematically in Figure 4 A.

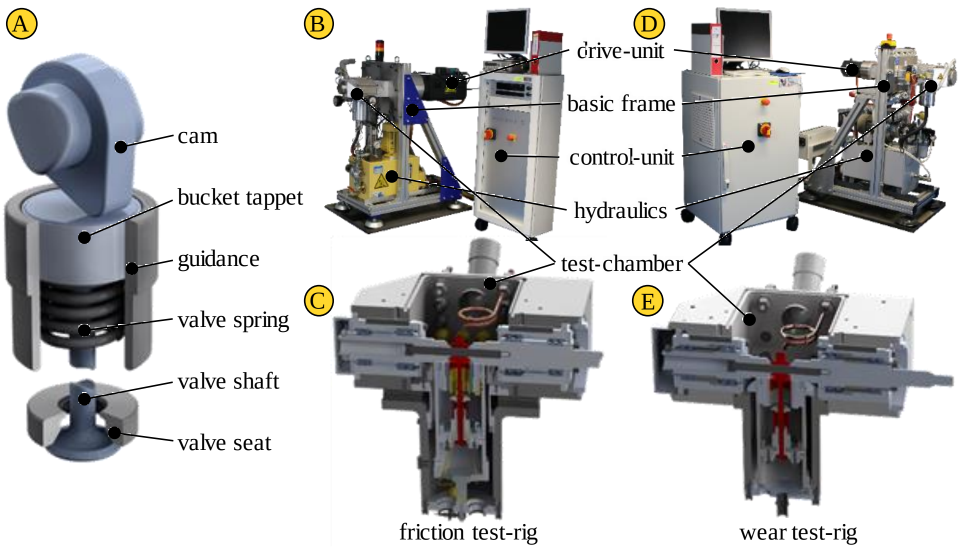


Figure 4. Tribological system of the cam/bucket tappet contact (A), design of the cam/bucket tappet friction (B, C) and wear (D, E) component test-rigs

Despite advantages of bucket tappets compared to roller tappets with regard to design and production costs or lightweight design aspects, there is still a need for improvement of bucket tappets with respect to frictional behaviour (Gutzmer, 2007). Latter can be achieved by the means of surface modification, such as microtexturing (Tremmel et al., 2017) or the use of tribological coatings (Dobrenizki et al., 2016). While a simulation based approach (Marian et al., 2017a) may help for a better understanding of local effects within the contact and shortening the development time, an experimental validation is mandatory. In general, tribological testing can range from complete technical systems under real operating conditions to lab-based tests with simplified test specimens. Component test-rigs allow a mechanism-oriented and reproducible study of numerous parameter combinations under fixed constraints in a time and cost efficient way while ensuring transferability into industrial application (Czichos and Habig, 2015). For these reasons, a one cam/one bucket tappet component test-rig with the aim of studying friction behaviour was developed (Figure 4 B, C). Later it was complemented by a second, modified test-rig in order to study wear behaviour (Figure 4 D, E).

4.2. Development of the cam/bucket tappet component test-rigs

The cam/tappet friction component test-rig, which was designed systematically following Pahl and Beitz (1996), is presented and documented in detail by Schulz (2013). Therefore, at first requirements were defined in the planning phase. In the conceptual phase, the essential functions to be fulfilled by the test-rig were compiled by abstraction. By means of functional factorization, sub-functions were derived and solution principles were found. A solution concept was generated, defined and then completed into a final design with modular architecture by combining and evaluating these principles. The overall design of the test-rig is divided into the modules test-chamber, basic frame, drive-unit, hydraulics and

control-unit (Figure 4 B). In the test-chamber, a single cam is in contact with a single bucket tappet. A cut through the test-chamber is shown in Figure 4 C. The cam is symmetrical with a cone seat on both sides and held by two shafts. The test-unit, in which tappet and valve are moving linearly, is mounted elastically at the bottom and supported by four crosswise arranged, preloaded piezoelectric force sensors at the top. Thus, the friction force within the contact can be determined. A lubricant unit with a temperature-controlled reservoir realizes the lubrication of the contact.
However, the complex measuring systems are not suitable for long-term tests due to complexity, sensitivity and high setup and maintenance costs. Nevertheless, long-term wear and fatigue of surface modified bucket tappets determine component lifetime and consequently need to be studied (Marian et al., 2017b). For that reason, a separate cam/tappet component test-rig for time-efficient wear testing was realized, see Figure 4 D. In general, the development was also based on the systematic approaches to design (Pahl and Beitz, 1996). However, the results of previously completed development phases were taken into account and adapted. In the first step, the existing design structure was analysed and selected sub-functions and solution principles were modified to the changed requirements while others were preserved. In the following steps, the modifications were further elaborated. This especially applies for the simplified test-setup and reduced measurement systems. A cut through the test-chamber is shown in Figure 4 E. For example, the friction force is no longer detected as a measurement value using piezoelectric force sensors. Instead, the friction behaviour is derived from the theoretical moment of resistance and the applied torque, which can be detected by a high-resolution torquemeter. This in turn allows a different, simplified design of the test-unit as certain features were no longer necessary, also enabling an easier setup and preparation process.

4.3. Ontology-based representation of the test-rig

Especially because of the fact that some features of the test-unit became obsolete to the new requirements for testing wear behaviour, the test rig is well-suited for finding and evaluating the introduced IF-mechanisms (Section 3.3). For this, the work results of the development process are pre-defined as a class system (Figure 5 on the left side). The relations follow the logical procedure of such processes. Therefore, functions are derived from requirements, while they are fulfilled by solution principles. These are realized by the assemblies, subassemblies and parts of the design phase. This structure is independent from the test-rig and therefore can be regarded as a basic structure. In the section of the test-rig ontology, shown in Figure 5, the classes and individuals at the bottom right were extracted automatically (Figure 5, dashed boxes).

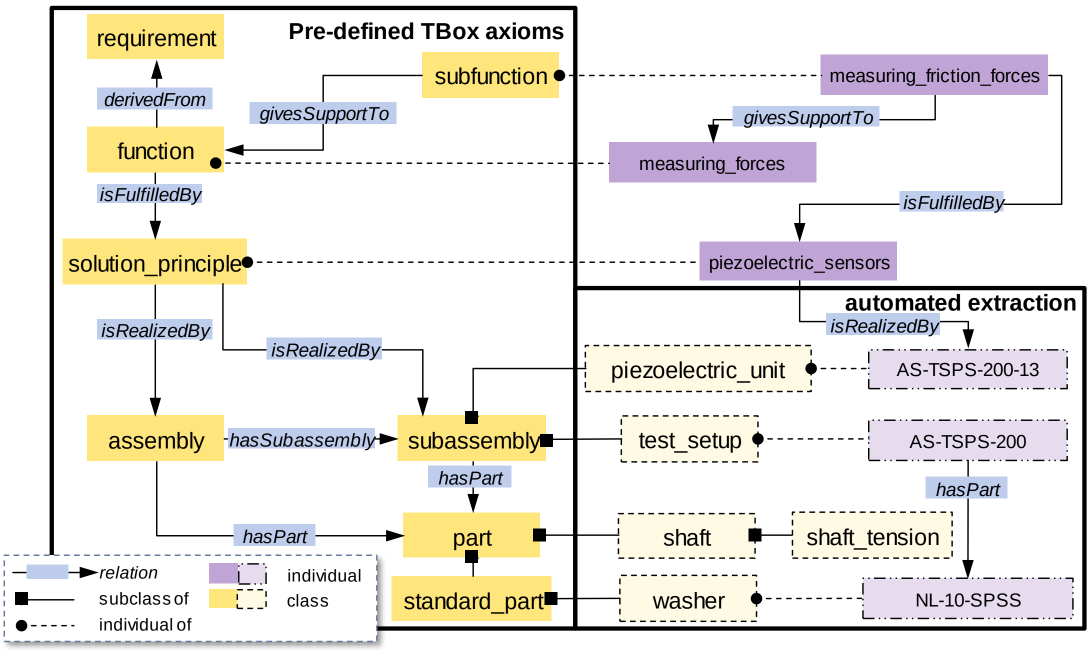


Figure 5. Example ontology with automated-extracted elements of the test-rig

Within the extraction not just subclasses of the pre-defined classes are added. The extracted class **shaft** is a subclass of **part**, while **shaft_tension** is a subclass of **shaft** and is also extracted automatically. This includes the possibility to define arbitrarily detailed class systems for specific use cases on the basic structure automatically. The individuals in Figure 5 are extracted as shown in Section 3.2. For instance, the individual **AS-TSPS-200** is of the type **test-setup** and therefore a **subassembly**. The individual for the standard part **washer NL-10-SPSS** is part of the test-setup of the test-rig, so the relation ***hasPart*** is set between the two individuals (Figure 5). The automated extraction is limited to the CAD representations of a product for now, so the early phases have to be integrated manually. For instance, the solution principle **piezoelectric_sensors**, which fulfils the subfunction **measuring_friction_force**, is realized by the individual **AS-TSPS-200-13**, which is of the type **piezoelectric_unit** (Figure 5). However, it is essential to connect the work results of the phases for supporting the reuse of development processes, thus impacts of changes, can be made traceable. The ontology provides a powerful way to establish such relations even the displayed contents are hold in different formats.

4.4. Testing queries in SPARQL Update

A *cascading forgetting* behaviour can be achieved by exploiting certain metaproperties like rigidity and dependency (Section 3.3). Among others, the classes **function**, **subfunction** and **solution_principle** (Figure 5), can be seen as rigid. In addition, the class **subfunction** depends on class **solution_principle** with respect to the relation ***isFulfilledBy*** (Figure 5) since for each individual of a **subfunction** there has to be also one of **solution_principle** to fulfil the subfunction. Similarly, the class **subassembly** depends on class **solution_principle**. The ABox for the test-rig contains, inter alia, the following axioms:

(1) **subassembly (AS-TSPS-200-13)**

(2) **solution_principle (piezoelectric_sensors)**

(3) **subfunction (measuring_friction_forces)**

(4) **function (measuring_forces)**

(5) ***givesSupportTo*** **(measuring_friction_forces, measuring_forces)**

(6) ***isFulfilledBy*** **(measuring_friction_forces, piezoelectric_sensors)**

(7) ***isRealizedBy*** **(piezoelectric_sensors, AS-TSPS-200-13)**

There are no other individuals belonging to class **solution_principle**, which are connected to **measuring_friction_forces** via the relation ***isFulfilledBy***. Now (2) is forgotten because the measurement with piezoelectric sensors is obsolete to the new test-rig. This is accomplished by the following SPARQL update query with an empty *WHERE* clause:

```
DELETE { piezoelectric_sensors a : solution_principle }
WHERE  {    }
```

The OntoClean-guided semantics introduced in Section 3.3 lead to the following result: The rigidity of class **solution_principle** results in the deletion of all axioms from the ABox containing individual **piezoelectric_sensors**, meaning that axioms (6) and (7) are also removed. Furthermore, axiom (3), is deleted, since the class **subfunction** depends on the **solution_principle** and there is no individual belonging to class **solution_principle** left, as mentioned afore. Furthermore, the deletion of (3) together with the fact that **subfunction** also constitutes a rigid class leads to the deletion of all axioms containing the individual **measuring_friction_forces**. This causes axiom (5) to be deleted. In addition, axiom (1) is deleted, since the **subassembly** depends on a **solution_principle** with respect to the relation ***isRealizedBy***. As illustrated, the OntoClean-guided semantics of SPARQL update lead to a cascading behaviour of forgetting: The deletion of axiom (2) triggers the deletion of other related axioms. Please note that due to the open-world semantics of description logics, the deletion of axiom (1), (3), (5), (6) and (7) does not logically follow from the deletion of (2). This cascading behaviour can only be achieved by the OntoClean-guided semantics of SPARQL update, which is used to implement this IF-mechanism.

4.5. Discussion

Based on the obtained results, it can be seen that the automated ontology generation from existing CAD models allowed an appropriate representation of the embodiment design phase of the test-rig. The manually effort was restricted to the implementation of the basic structure, thus existing CAD models can be added easily to this pre-defined class system. To achieve a holistic support of the product development process, automated solutions for the integration of early phases are part of future research. The integration of the different work results is one step, another challenge is to automatically relate the development phases, which is also part of current and future research. Moreover, it was shown, that the IF-mechanism cascading forgetting led to the deletion of further axioms. The deletion of the piezoelectric sensors, which was declared as a solution principle, affected the concerning subfunction and even the subassembly piezoelectric unit. In consequence, all elements, which are connected to the obsolete solution principle, were removed, thus white spaces for new ideas are available. This provides a great support for the reuse of the test-rig-model in the new development task, because with the support of the IF-operator just one removal causes the forgetting of all of the obsolete elements. Thus, the time-consuming and error-prone manual searching becomes redundant and provides more time for creative and generating development tasks.

5. Conclusion and future work

While the earlier phases have to be implemented manually within the ontology, a reliable approach was shown to automate the transfer of CAD data into OWL. However, the high effort of the manually implemented parts in ontologies requires automated solutions for the earlier phases. A software interface between the requirements management solution DOORS and the OWL representation can automate the inclusion of requirements in the representation and is part of future research. The future work focuses on an approach called Enterprise Information Integration (EII), which enables storing and synchronizing data and information held in heterogeneous formats. Moreover, another test-rig is currently developed and manufactured. For latter, the original tribological system is further abstracted into a model ring/disk tappet system in order to gain further insights of fundamental effects and phenomena. Again, the development process of the model test-rig is based on reuse and adaptation from the earlier developed test-rigs. Therefore, IF-mechanisms can also be applied and thus further evaluated. Beside this, the implementation of further introduced IF-mechanisms is the subject of current research.

Acknowledgement

The authors gratefully acknowledge the financial support of project WA 2913/22-1 and STA 572/15-1 within the Priority Program 1921 as well as project WA 2913/3-2 within the Priority Program 1551, by the German Research Foundation (DFG).

References

Ahmeti, A; Calvanese, D; Polleres, A.; Savenkov, V. (2016), “Handling inconsistencies due to class disjointness in SPARQL updates”, *The Semantic Web. 13th International Conference, Heraklion, Greece, May 29 - June 02, 2016*. Springer, Cham, pp. 387-404. doi: 10.1007/978-3-319-34129-3

Albers, A.; Bursac, N.; Wintergerst, E. (2015), "Produktgenerationenentwicklung – Bedeutung und Herausforderungen aus einer entwicklungsmethodischen Perspektive", *Beiträge zum Stuttgarter Symposium für Produktentwicklung (SSP2015), Stuttgart, Germany, 19.06.2015*, Fraunhofer IAO, Stuttgart

Baader, Franz; Horrocks, Ian; Lutz, Carsten; Sattler, Uli (2017): An Introduction to Description Logic, Cambridge university press, Cambridge. doi: 10.1017/978-1139-0253-5-5

Czichos, H. and Habig, K.-H. (2015), Tribologie-Handbuch. Tribometrie, Tribomaterialien, Tribotechnik, Springer Fachmedien, Wiesbaden, Germany, doi: 10.1007/978-3-8348-2236-9

Dobrenizki, L.; Tremmel, S.; Wartzack, S.; Hofmann, D. C.; Brögelmann, T.; Bobzin, K.; Bagcivan, N.; Musayev, Y.; Hosenfeld, T. (2016), “Efficiency improvement in automobile bucket tappet/camshaft contacts by DLC coatings – Influence of engine oil, temperature and camshaft speed”, Surface and Coatings Technology, Vol. 308, pp. 360–373, doi: 10.1016/j.surfcoat.2016.09.041

Gearon, P.; Polleres, A.; Passant, A., SPARQL 1.1 update. W3C recommendation, W3C, Mar. 2013. https://www.w3.org/TR/2013/REC-sparql11-update-20130321/.

Gruber, T.R. (1993), "A Translation Approach to Portable Ontology Specifications", *Knowledge Acquisition*, Vol. 5 No. 2, pp.199-220.

Guarino, N. and Welty, C.A. (2009): “An overview of OntoClean”, *Handbook on Ontologies, International Handbooks on Information Systems*, pp. 201-220. Springer, Heidelberg doi: 10.1007/978-3-540-92673-3

Gutzmer, P. (2007), “Weniger Reibung – Schlüssel für mehr Effizienz”, Motortechnische Zeitschrift, Vol. 68 Nr. 4, p. 243

Kara, S. (2015), *Vergiss es!*. [online] Die Zeit online. Available at: http://www.zeit.de/2015/33/vergessen-gedaechtnis-entwicklung-gesellschaft-beziehungen-psychologie (accessed 02.11.2017).

Kestel, P.; Luft, T.; Schon, C.; Kügler, P.; Bayer T.; Schleich, B.; Staab, S.; Wartzack, S. (2017), “Konzept zur zielgerichteten, ontologiebasierten Wiederverwendung von Produktmodellen”, *Beiträge zum 28. DfX-Symposium, Bamberg, Germany, October 04-05, 2017,* TuTech Verlag, Hamburg, pp. 241 – 252.

Kienreich, W. and Strohmaier, M. (2006), “Wissensmodellierung-Basis für die Anwendung semantischer Technologien”, In: Pellegrini, T. and Blumauer, A.(Ed.), *Semantic Web,* Springer, Berlin Heidelberg, Germany, pp. 359-371. doi: 10.1007/3-540-29325-6

Koch, F.; Geiger, U. (1997), “Reibungsanalyse der Kolbengruppe im gefeuerten Motorbetrieb”, Tribologie und Schmierungstechnik, Vol. 44 No. 3, pp. 109–110.

Marian, M.; Weschta, M.; Tremmel, S.; Wartzack, S. (2017), “Simulation of Microtextured Surfaces in Starved EHL Contacts Using Commercial FE Software”, Materials Performance and Characterization, Vol. 6 No. 2, pp. 165 – 181, doi: 10.1520/MPC20160010.

Marian, M.; Zahner, M.; Tremmel, S.; Andreas, K.; Merklein, M.; Wartzack, S. (2017), „Design, Manufacturing and Tribological Performance of Microtextured Bucket Tappets for Friction Reduction in the Valve Train”, *6th World Tribology Congress, Beijing, China, September 17 – 22, 2017*

Pahl, G. and Beitz, W. (1996), Engineering Design: A Systematic Approach, Springer, Berlin. doi: 10.1007/978-1-4471-3581-4

Rude, S. (1998), Wissensbasiertes Konstruieren, Habilitation, Universität Fridericiana (TH) Karlsruhe.

Schon, C.; Staab, S. (2017), Towards SPARQL Instance-Level Update in the Presence of OWL-DL TBoxes, *Proceedings of the Joint Ontology Workshops 2017 The Tyrolean Autumn, Bozen, Italy, September 21 - 23, 2017*

Schulz, E. (2013), Wissensbasierte Vorhersage der Reibung im komplexen tribologischen Systemen am Beispiel des Kontakts Nockenwelle/beschichteter Tassenstößel, PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Seaborne, A. and Harris, S. (2013), SPARQL 1.1 query language. W3C recommendation, W3C, Mar. 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

Tremmel, S.; Marian, M.; Zahner, M.; Weschta, M.; Engel, U.; Wartzack, S.; Merklein, M. (2017), „Reibungsreduzierung in EHD-Kontakten durch mikrostrukturierte Bauteiloberflächen – Auslegung, Gestaltung und umformtechnische Herstellung“, 58. Tribologie-Fachtagung. Reibung Schmierung und Verschleiß. Forschung und praktische Anwendungen, Gesellschaft für Tribologie e. V., Aachen, Sonderband Abschlusskolloquium „Ressourceneffiziente Konstruktionselemente“ SPP 1551 der DFG, pp. 149–166

VDI 5610 (2009), Wissensmanagement im Ingenieurwesen, Beuth, Berlin.

[Corresponding author data - this will be inserted automatically]
[---]
[---]
[---]
[---]
[---]
[Do not delete or modify the author's data area]