Ontological Semantics of Standards and PLM Repositories in the Product Development Phase

Marco Franke[1], Patrick Klein[1], Lutz Schröder[2], Klaus-Dieter Thoben[1]

Abstract In order to optimally exploit the large amounts of engineering information stored in contemporary PLM systems, the concept of knowledge based engineering (KBE) can be considered from a PLM perspective. By eventually combining product structures and implicit semantics provided by PLM-systems on the one hand, and domain-specific standards on the other hand we believe to have identified a key enabler KBE.
As an initial step we describe a coupling of a CAD system with a semantic representation of engineering knowledge using formal ontologies. By application of automatic reasoning, engineering knowledge gained from the product structure and domain-specific standards allows us to reduce time-consuming manual work in classifying overlaps between parts in a CAD model as intentional overlaps (e.g. with gaskets) or design failures.

Keywords PLM, KBE, Semantics, Ontology, Reasoner

1 Introduction

Today, business competitiveness is usually broken down into success factors such as decreased time-to-market, higher success rates in product introduction, reduced project failure rates, minimized manufacturing costs, increased product and process innovation, and improved communication among departments and business partners. This obviously impacts the requirements on classical business applications like ERP (Enterprise Resource Planning), PLM, or CAx software, and consequently affects the corresponding research activities [1].

Even the performance of the initial product development phase is affected not only by technological challenges but also by the socio-technical context in which it happens. A scenario of a globally distributed development team may serve to illustrate this. In this scenario, networked enterprise systems or PLM-systems, respectively, become the main backbone for coordinating geographically dispersed engineering activities [2].

In fact, collaborative features have become standard in contemporary PLM systems, such as ENOVIA [3], providing a single front-end to multiple information sources, enabling dispersed data storage, real time visualisation of the emerging product, global change management, or design-in-context approaches.

[1] BIBA, Hochschulring 20, 28213 Bremen, Germany
email: klp@biba.uni-bremen.de, fma@biba.uni-bremen.de, tho@biba.uni-bremen.de
[2] DFKI, Cartesium Enrique-Schmidt-Straße 5, 28359 Bremen, Germany
email: Lutz.Schroeder@dfki.de

Nevertheless, as pointed out by Garcia & Fan [2] one practical question is still not solved sufficiently: “How to retain and capitalise the large amount of engineering information stored in PLM repositories as intellectual property assets?” [2]

This leads to the field of knowledge based engineering and in detail to research covering KBE services within PLM [4,2,5]: Keeping in mind that PLM-systems provide a key technology that enables generic and cost-effective sharing of product and process information across a wide range of software systems (not only CAx) and across organizational barriers, several researchers have raised the idea of implementing standardised PLM interfaces as a possible solution for interoperability between two different KBE-systems [4]. As one of the results the ‘KBE Services for PLM’ RFP was published in September 2005 by OMG [6].

However, as we discuss later, standardisation in general is not only a possible solution for such an interoperability issue. In combination with specific PDM/PLM information, standards can play an important role in covering one of the most critical KBE issues: knowledge acquisition.

2 Background

The high impact of product-related decisions in the initial development phase on the overall product costs and lead time is as well-known as the coexistence of a pronounced lack of product-related knowledge in this phase.

While some current research approaches try to decrease lead-time by shifting the identification and solving of engineering problems to the early phase of the product development process (so called *front loading)* [7], others are addressing solutions to ramp up the initial creative phases by specific supporting tools *(inventive design)*. The approach of knowledge based engineering directly focuses on the reduction of lead-time and costs by supporting and in particular automating repetitive design tasks [8].

Our own qualitative experience in the area of knowledge management gained in research projects in collaboration with several industry branches (aircraft, maritime, automotive) indicates that such tasks represent most of the work in the product development process. According to, e.g., a quantitative analysis by Skarka [8], a proportion of about 80% of the overall design tasks is routine and consists of repetitive tasks such as adaptation of existing parts to slight changes in the overall geometry, or checking for clashes and omissions.

The enormous potential of a successfully implemented KBE solution has been already validated by several research projects [9], [10], [11]. By each of those implementations, a notable time reduction from several days to a few hours for the respective design tasks has been achieved, while in parallel a constant quality due to the repeatability can be ensured.

However, this is by no means a general justification for an unlimited deployment of a KBE system. A usage of KBE technologies may not be effective in different situations, e.g. if a problem is simple enough to solve it in a less technology-centered way (i.e. without KBE technologies) or if it is not possible to extract or to codifiy the required knowledge, e.g. in the absence of a clearly defined design process [8].

2.2 Different approaches to knowledge based engineering

As already pointed out by Penoyer [12], knowledge based engineering appears, at first glance, to be a tautology – usually every person (and especially every engineer) involved in a product development process will define her engineering tasks as based on specific knowledge.

Hence for our purposes, knowledge based engineering (KBE) will be defined in close conjunction with KBE systems. Within a KBE system, design knowledge is represented in a formal manner and enables the system to automate specific design tasks mostly unique to the company's product development experience.

Each KBE system provides on the one hand an interface to capture the knowledge in terms of logical rules, algorithms, or constraints, and on the other hand an output module to trigger adjacent CAx systems or/and visualise results [13].

In this sense, knowledge based engineering can be seen as the process of gathering, managing, and using engineering knowledge to automate the design process by usage of a KBE system [14]. In this context, the meaning of *automate* even covers analysis tasks in terms of validation or quality checking, since the interpretation of the output of CAx tools, such as CATIA's DMU Space Analysis, requires engineering knowledge about the mechanical parts involved.

An emerging trend in the field of knowledge based engineering is to set up a background ontology, link one or more of the available CAx engineering tools to it, and thus provide context specific engineering knowledge for different tasks covered by separate CAx tools [11]. Other research addresses the idea of using the ontology in order to represent a generative model and thus enabling design automation [8].

Surprisingly, one of the most noticeable advantages of such an approach seems to be not yet fully exhausted by the solutions developed so far: the ability of using formal logic and automated reasoning in order to generate further findings and reports for control and steering purposes.

A further advantage of the usage of ontologies appears in the context of the upcoming requirement for PLM systems to capture and manage the technical decisions made by product developers in the initial development phase. Such a decision-tracking is of increasing importance in the context of product warranties on the one hand, and as a valuable input for follow-up product developments on the other hand.

The standard approach to retaining product design related knowledge and experience is to produce and store documents such as lessons-learned or best-practices.

Consequently, the respective expert defines the terminology, verbalisation, and level of detail of the represented knowledge by herself. In the long run, this way of archival storage implies a continuous decrease of comprehensibility, since terminology and wording may change over time. The transfer of knowledge into an ontology expressed in a description logic with a formally grounded semantics avoids such a semantic dilution and thus ensures that the codified knowledge is sustainable, in particular remains readable, maintainable, and convertible over time.

2.3 The challenge of knowledge acquisition

The requirement of capturing domain specific knowledge can be seen as one of the main challenges in the field of Knowledge Based Engineering [15].

Even if several methodologies (e.g. MOKA [16]) have been elaborated to guide knowledge acquisition activities and thus avoid omitting essential knowledge [8], they usually require a time-consuming collection and analysis of (often implicit) knowledge about the product and its design process, respectively [17]. Thus, most approaches to designing KBE-Tools address especially repetitive engineering tasks [18,10], since the potential to reduce time and cost by means of such approaches has to be balanced against the effort needed to gather and formalize the required knowledge in a scheme (e.g. an ontology) [18].

Contemporary CAD systems provide several enhancements to support product data management features, and thus very often constitute the main link to a global PLM-system within an enterprise IT infrastructure. These modules allow not only storing and managing a broad range of product-related non-geometrical data, but give the user a visual and intuitive access via the graphical representation of a product and its product structure, respectively [19]. Thus, capturing PDM-data via context specific dialogs within the respective CAD-systems has become common practice.

Based upon these coupling concepts, the use of a CAD user interface for a KBE system is an obvious and already implemented idea. In fact, many of the leading CAD applications provide add-on modules for KBE related features. The *knowledge advisor*, *knowledge expert* and *product knowledge template* modules of the CAD application CATIA can serve as examples. Based on a parameterized CAD model, they provide functions like formulas (to create dependencies between parameters), rules (such as If… then...) and power copys (user defined features, allowing to partly reuse design procedures) [20]. Nevertheless, integrated methods for an easy knowledge acquisition remain a key hurdle for the application of these functions [8].

2.3 PLM and standards - an underestimated source for knowledge acquisition

For PLM systems, *product structures* have become one of the most important backbones to which the various types of metadata are attached. Within PLM applications, the requirements of taxonomical *naming* and *numbering* lead to sophisticated algorithms that cope with the complexity of providing a distinct, non-redundant namespace [21]. In parallel to such internal representation logic, several formal standards are used in the area of PLM in order to represent the product and its product structure appropriately.

In the area of mechanical engineering, standardisation is usually not only a clustered set of generic product information, or a taxonomy of a specific domain, but it comprises a high amount of codified knowledge, in terms of, e.g., calculation rules, engineering constraints, schemes for data exchange etc. The use of standards to cover such codified knowledge is based on a long history in the field of mechanical engineering, ranging from the VDI 2230 guideline that treats the systematic calculation of high duty bolted joints [22] up to the ISO 10303 standard for the computer-interpretable representation and exchange of product manufacturing information. Several specific KBE solutions cover the idea of using such codified knowledge for a specific design problem - a good example is given by [23], which implements the Italian VSR/PED rules for the verification of pressure vessels.

By a combination of both types of knowledge – product structures and namespaces provided by PLM systems, and existing domain specific standards – we believe to have identified a key stepping stone to harnessing knowledge acquisition in a principled and sustainable way.

As an initial proof of concept for the benefits that can be achieved using this type of combination, we describe below a semantic analysis of clashes and overlaps in CAD files. Our prototype of an analysis tool (called OntoDMU) is able to check semantically if an overlapping is a design failure or an intended feature, at least in those cases where standard parts are involved.

Specifically, we exploit that when a standard part is used in a product, the respective standardisation identifier remains available, usually as a section of the item name in the CAD model. For example in CATIA V5, when a nut is chosen from the standard part catalogue of the application, an expression such as ***ISO 4034*** *NUT M14 STEEL GRADE C HEXAGON HEAD NONPREFERRED* will be provided as a default part name in the product structure. This enables us to connect the relevant standard (in this case, ISO 4034) with a background ontology, which in turn helps us interpret the output of the analysis tool.

3 Practical benefits drafted in a Sample Scenario

Validating the correctness (the so-called quality) of a CAD model by analysing its compliance to corresponding engineering knowledge can be seen as a typical job for a designer. In this context, contemporary CAD files provide her with several support modules, e.g. for validating a mock-up against assembly requests, or checking its conformance with the PLM namespaces. One of those tasks is an investigation of the CAD-model in order to distinguish between intended part overlaps and overlaps to be attributed to design failures.

Looking specifically at the case of overlaps, it is by no means the case that every overlap is actually a design error — e.g. overlaps are often intentional in the case of bolts, whose threads are typically not modelled in the CAD software, so that a bolt will overlap with its nut. Similarly, deformations of gaskets (e.g. O-rings) are typically ignored (both for computational reasons and because one wishes to have the undeformed shape of the gasket in the design, e.g. for purposes of exploded views) so that they overlap with adjacent parts, even if sizes are appropriate. In fact, overlaps are actually mandatory in both examples, but do of course represent design errors in other cases, some of them subtly different – e.g. a bolt should not overlap with the parts it connects unless the latter also have threads.

Picking up the above mentioned gasket example Figure 1 shows a half section view of the 3D-CAD-model of two flanges screwed together (e.g. used in context of pipe coupling).

The small circle represents a gasket. The parts in the background represent a bolt and a nut - screwed together. By using a half-section view of the assembled parts, a mechanical designer can check the correctness of the design and the CAD-model respectively (position, dimensions, overlapping etc). Thus, not only the gasket's position in the flange notch becomes visible, but also its intersection with the flange.

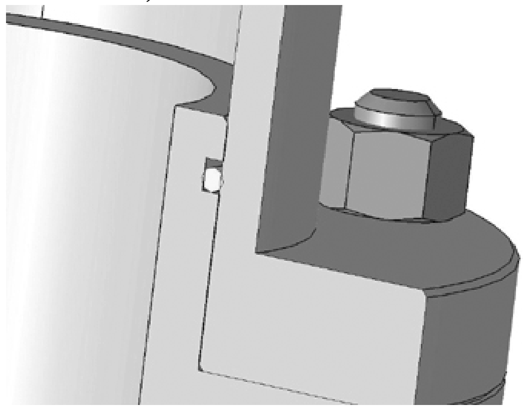
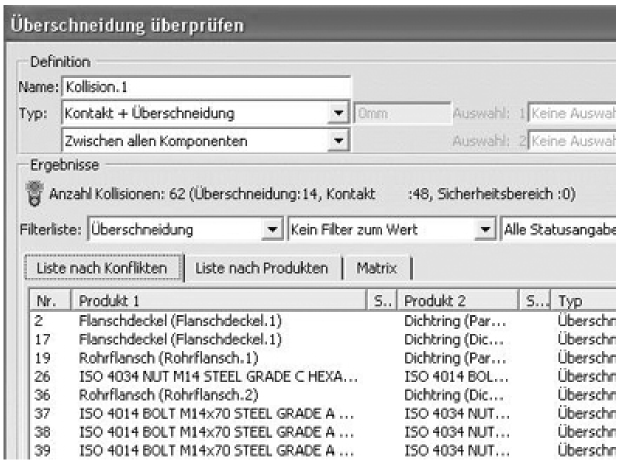


Fig. 1 Half-section view of the assembled flange

Being aware that a gasket normally consists of deformable Flouride rubber (FPM) the mechanical designer can easily identify the correctness of the overlap and the assembly as a whole, since no overlap would lead to a leaky assembly. Unfortunately, CAD-models can become confusing for complex products. To check overlaps of a gas-tanker assembly-model, for instance, leads to thousands of gasket intersections.

Using an interference detection module such as CATIA's DMU Space Analysis will provide the mechanical designer with a complete list of all overlaps, but the tool cannot distinguish between required overlaps and unintentional clashes. This is caused by the fact that no inferences are possible from a geometrical representation of a part to the part itself (for example: In a CAD application there is absolutely no difference between a geometrical model of a ring and a geometrical model of a gasket).



Überschneidung überprüfen

Definition

Name: Kollision.1

Typ: Kontakt + Überschneidung | 0mm | Auswahl: 1 Keine Auswah

Zwischen allen Komponenten | Auswahl: 2 Keine Auswah

Ergebnisse

Anzahl Kollisionen: 62 (Überschneidung:14, Kontakt :48, Sicherheitsbereich :0)

Filterliste: Überschneidung | Kein Filter zum Wert | Alle Statusangabe

Liste nach Konflikten | Liste nach Produkten | Matrix

Nr.	Produkt 1	S..	Produkt 2	S...	Typ
2	Flanschdeckel (Flanschdeckel.1)		Dichtring (Par...		Überschn
17	Flanschdeckel (Flanschdeckel.1)		Dichtring (Dic...		Überschn
19	Rohrflansch (Rohrflansch.1)		Dichtring (Par...		Überschn
26	ISO 4034 NUT M14 STEEL GRADE C HEXA...		ISO 4014 BOL...		Überschn
36	Rohrflansch (Rohrflansch.2)		Dichtring (Dic...		Überschn
37	ISO 4014 BOLT M14x70 STEEL GRADE A ...		ISO 4034 NUT...		Überschn
38	ISO 4014 BOLT M14x70 STEEL GRADE A ...		ISO 4034 NUT...		Überschn
39	ISO 4014 BOLT M14x70 STEEL GRADE A ...		ISO 4034 NUT...		Überschn

Fig.2 Space Analysis report - screenshot

Figure 2 is a screenshot of the DMU Space Analysis report belonging to the CAD-model shown in Figure 1. Even if it is a quite simple product and only identified overlappings are displayed, the list gets quite long and leads to time-consuming manual work.

By using the OntoDMU tool for the ontological analysis of the output of the DMU analyser as described in the present work, however, the designer can analyze this list of overlaps semantically and identify those overlaps that are not allowed.

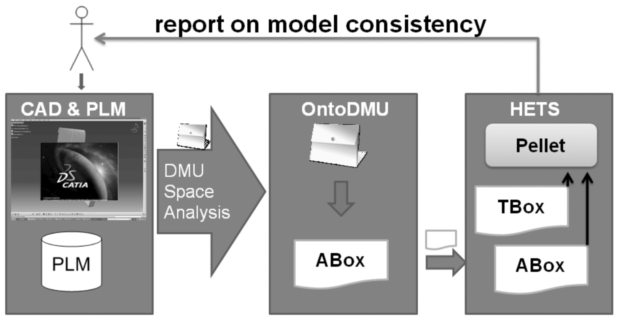


Fig. 3 architecture of the initial prototype

As shown in Figure 3, the OntoDMU prototype transforms the output of the DMU Space Analysis module into a set of individuals set against a background ontology, thus making it available for semantic analys using state-of-the-art automated reasoning. Next, we proceed to describe details of this method.

4 Approach

To capture the semantics of standards and PLM repositories, we propose to make use of formal ontologies expressed in a formal ontology language at the level of so-called description logics; specifically, we use the standard ontology language OWL-DL (Web Ontology Language), a W3C recommendation [24]. Description logics are tuned to offer an optimal degree of expressive power while retaining efficient decidability, and indeed come with high-performance optimized reasoners such as Pellet [25]. For purposes of describing engineering designs, this means that our background ontology is able to describe simple relationships between parts and components, such as existence, parthood, cardinality etc., but not the geometry or topology of a model. However, it turns out that a surprisingly large amount of knowledge can be captured in such a simple framework, and exploited for the automated consistency checking of CAD models. In t his process, it is precisely the simplicity of the language that allows us to use efficient reasoning and thus achieve a practically feasible semantic framework, which in the end even does offer support for geometry-related issues, such as overlapping, on a suitable level of abstraction. We emphasize that a representation in a formal logic carries a number of advantages over a hard-wired representation in software, in particular

- increased clarity of the representation
- independence of accidental features of the software environment
- reduced likelihood of errors, due to simplicity of expression and absence of side-effects
- improved interoperability.

It turns out that in order to reduce the complexity of modelling and keep an optimal level of modularity, it is useful to maintain two types of ontologies: An ambient ontology that covers abstract engineering knowledge, such as that rubber is deformable (and therefore rubber parts may overlap with adjacent parts since the deformation is usually not explicitly modelled) and, embedded therein, an ontology of standard (or enterprise standard) parts which represents and classifies a part catalogue against the ambient ontology, but typically does not otherwise encode any background knowledge. One benefit of this approach is that both parts of the ontology become much easier to maintain, and in particular the ontology of standard parts can mostly be generated automatically from part databases in the CAD system.

4.1 Using a background ontology

As discussed above, an ontology expressed in a formal description logic allows one to formally represent and store domain specific knowledge. It enables in particular a perspective where we regard a CAD model as a collection of instances of generic objects that we can view against the backdrop of the ontology. The ontology then serves as a template for maintaining consistency during the development of a product or the creation of variants. Moreover, the designer can further develop the ontology in order to make domain knowledge assumptions explicit and facilitate reuse of his designs.

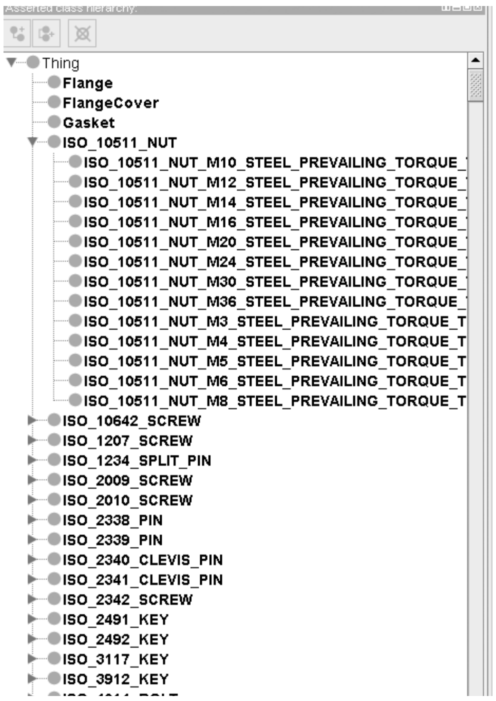
We briefly recall some of the basic concepts of OWL to facilitate the understanding of the examples given further below. An OWL *class* represents a collection of objects; e.g. the class ‘bolt’ stands for the collection of all individual bolts. Similarly, a *property* represents a relationship between objects, such as parthood. From the basic classes, one forms *concepts* by applying Boolean operators as known from propositional logic (conjunction, disjunction, negation) and so-called *restrictions* which govern the way in which an object is expected to be related to other objects. E.g. an existential restriction on the property ‘hasFeature’, qualified by the class ‘thread’, designates all objects that have some feature that is a thread. Similarly, a universal restriction on the property ‘hasPart’, qualified by the class ‘standardPart’, designates objects composed only of standard parts.

An ontological knowledge base then consists of two parts offering different perspectives on the domain: The structural information of a domain is characterized through its TBox (the *terminology*). The TBox consists of a set of inclusions between concepts, and as such allows expressing general knowledge such as ‘every bolt has a thread’, or ‘every car has four wheels and a colour’. Contrastingly, the ABox (the *assertions*) contains knowledge about individuals, say a particular car or a given occurrence of a standard part in a CAD model. It can state either that a given named individual (say, ‘myCar’) belongs to a given concept (e.g. that myCar is, in fact, a car) or that two individuals are related by a given property (e.g. that myCar is owned by me).

By default, an ontology has no restriction for naming. For this reason, the same element can have different labels in two or more ontologies (precisely because OWL does not implement the so-called unique-name assumption), but would not be detected as the same in case of merging the ontologies. Therefore, it is desirable that the label of each element is kept unique. If an element has a unique name in the real world, a designer can achieve such a unique labelling by using the same name within the ontology, following an appropriate transformation of name spaces. Additionally, an encapsulation into name spaces can be used to ensure unique labelling of items.

As already indicated, using an ontology as part of a KBE solution can improve the engineering processes, but at the same time, the modelling of an ontology can become very complex, especially if a generic approach is envisaged.

In the scenario described above, we focus on the standard part catalogue of CATIA V5 R16. This version comprises about 8838 standard parts, and each part has its specific properties and restrictions, which have to be implemented in the ontology (Figure 4). For this reason, one of the main challenges is to reduce the effort and the complexity of modelling. In order to avoid such time consuming manual work, we have foreseen a special function in OntoDMU to import standard parts into concepts of the ontology (refer to Section 4.4).



```
Thing
    Flange
    FlangeCover
    Gasket
    ISO_10511_NUT
        ISO_10511_NUT_M10_STEEL_PREVAILING_TORQUE_
        ISO_10511_NUT_M12_STEEL_PREVAILING_TORQUE_
        ISO_10511_NUT_M14_STEEL_PREVAILING_TORQUE_
        ISO_10511_NUT_M16_STEEL_PREVAILING_TORQUE_
        ISO_10511_NUT_M20_STEEL_PREVAILING_TORQUE_
        ISO_10511_NUT_M24_STEEL_PREVAILING_TORQUE_
        ISO_10511_NUT_M30_STEEL_PREVAILING_TORQUE_
        ISO_10511_NUT_M36_STEEL_PREVAILING_TORQUE_
        ISO_10511_NUT_M3_STEEL_PREVAILING_TORQUE_T
        ISO_10511_NUT_M4_STEEL_PREVAILING_TORQUE_T
        ISO_10511_NUT_M5_STEEL_PREVAILING_TORQUE_T
        ISO_10511_NUT_M6_STEEL_PREVAILING_TORQUE_T
        ISO_10511_NUT_M8_STEEL_PREVAILING_TORQUE_T
    ISO_10642_SCREW
    ISO_1207_SCREW
    ISO_1234_SPLIT_PIN
    ISO_2009_SCREW
    ISO_2010_SCREW
    ISO_2338_PIN
    ISO_2339_PIN
    ISO_2340_CLEVIS_PIN
    ISO_2341_CLEVIS_PIN
    ISO_2342_SCREW
    ISO_2491_KEY
    ISO_2492_KEY
    ISO_3117_KEY
    ISO_3912_KEY
```

Fig. 4 Detail of the background ontology – screenshot

The overall approach to determining the consistency of a CAD model with respect to the ontology is, then, as follows. The background ontology together with the ontology of standard parts exported from CATIA V5 formally constitutes a TBox. A second function of our OntoDMU tool is able to convert the output of CAD tools such as the DMU analyser into an OWL ABox over this TBox; then, the consistency check amounts to checking the consistency of the combined knowledge base. As discussed below, the technical framework that integrates all these tasks is the Bremen heterogeneous tool set Hets.

4.2 Ontology languages

We digress briefly to discuss our choice of ontology language, limiting ourselves to the three main sublanguages of OWL: OWL Lite, OWL DL and OWL Full [24].

OWL Full features the highest expressivity; however, it does not currently have efficient reasoning support, and the logical complexity of the language makes it unlikely that such support will be developed in the foreseeable future. Since our approach to consistency checking of CAD models relies crucially on fully automated reasoning, OWL Full is, thus, not a suitable option. As mentioned above, efficient reasoners do exist for the sublanguages OWL Lite and OWL DL [26]. The complexity of OWL Lite is markedly below that of OWL, so that more efficient reasoning is possible for ontologies limiting themselves to the expressive means of OWL Lite (and efficiency remains an issue in our framework, as both the output of the DMU analyser and the imported ontology of standard parts tend to become large rather quickly). However, the expressive power of OWL Lite turns out to be too limited for our purposes; in particular, OWL Lite excludes conjunction and universal restriction, which we need to say things like ‘bolts have threads *and* intersect only with nuts’ or ‘all member of the concept ***ISO 4034*** *NUT M14 STEEL* have an identical a diameter of 14mm.

Technically, OWL-DL ontologies can be written and stored in several ASCII-based formats. These formats can be translated into each other. Some reasoners, such Pellet, have corresponding translation functions. For using OWL-DL within Hets, it is necessary to generate the ontology in OWL Manchester Syntax [27]. Such a file can be opened with all common OWL readers, such as Protégé in version 4.

4.3 The Bremen heterogeneous tool set

We embed our background ontology as well as our interface tool into the Bremen heterogeneous tool set (Hets) [28] which allows for the integrated use of a wide variety of logics and associated analysis and reasoning tools in a common framework, accessed via a graphical interface and connected by a network of logic translations. Relevant for purposes of the present work are the support offered in Hets for ontology languages including in particular OWL-DL Manchester Syntax and, as a more expressive correspondence language, first order logic, as well as the facilities provided in the Hets implementation framework for the easy integration of further logics.

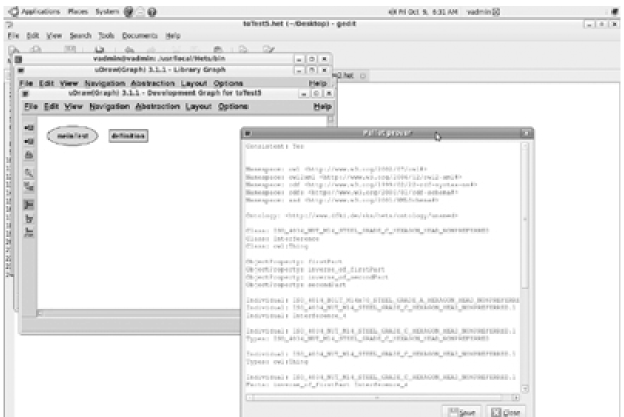

Applications Places System
vadmin
toTest5.het (~/Desktop) - gedit
File Edit View Search Tools Documents Help
vadmin@vadmin: /usr/local/Hets/bin
uDraw(Graph) 3.1.1 - Library Graph
File Edit View Navigation Abstraction Layout Options Help
uDraw(Graph) 3.1.1 - Development Graph for toTest5
File Edit View Navigation Abstraction Layout Options Help
definition
Save
Close
Plain Text
Tab Width: 8
Ln 2, Col 11
INS

Fig. 5 Hets graphical interface (screenshot)

The latter has allowed us to cast the output format of the DMU Analyzer as a very simple-minded logic, thus enabling integration of the OWL translation tool into the Hets framework and thereby, e.g., direct reasoning support for the combination of the tool output and the OWL background ontology in Pellet [25]. Figure 5 shows a screenshot of the Hets graphical interface and an interface window for a call to the Pellet reasoner.

4.4 Semi-automatic generation of Ontologies

The tool OntoDMU generates the ABox automatically and the background ontology semi-automatically. In the following, we describe these generation processes in more detail.

The background ontology should define all concepts a user needs for his own modelling purposes. In our concept, this includes standard parts (taken from the standard part catalogue of CATIA V5) as well as non-standard parts. Since non-standard parts are user or enterprise specific, the idea of importing non-standard parts is a priori a non-trivial proposition. We intend to implement an interface in order to transfer the product-structures and namespaces from PDM/PLM into the background ontology.

The standard parts, on the other hand, are stored in a separate catalogue-folder managed by CATIA. Hence OntoDMU can access the respective information without starting CATIA, and every change in the catalogue folder can be easily updated in the ontology. OntoDMU extracts all relevant information from the files automatically and transfers it into the ontology. A part from the standard catalogue is transformed into a concept, and its properties are inserted as a combination of data and object properties.

For example, the class of nuts ***ISO 4034*** *NUT M14 STEEL GRADE C HEXAGON HEAD NONPREFERRED* is such a catalogue part. Its name directly contains information about its properties. In this example, the material and the diameter of the nut can be read off from the name, and inserted as explicit properties of the item. Further information gained from the part name is the relevant standard (in this case, ISO 4034), which induces further properties by relations with the background ontology. In our case the respective information is: *every ISO 4034 part is a nut and as such has an inner thread.* In detail, this information arises as follows: we record in the ontology of standard parts that every ISO 4043 part is a nut, and we have captured, in the background ontology, the piece of general engineering knowledge stating that every nut has an inner thread.

As examples of ‘non-standard’ parts (i.e. not from the CATIA V5 catalogue), to be though of as enterprise standard, the ontology of our scenario includes classes *Flange*, *Gasket* and *FlangeCover*, which are currently maintained manually.

General knowledge about parts as such is then integrated with knowledge relating to the topic covered by our target geometric analysis tool, overlaps or, in the terminology used in the tool output, *interferences*, between parts. As discussed later, interferences may either be intentional or indicate design failures. The knowledge used in classifying interferences accordingly is modelled using properties of a dedicated class ‘interference’; details are given further below.

4.5 Structural information in the background ontology

The standard parts share a common interface of object properties. At present, OntoDMU can identify *type*, *material*, *dia-*

meter and *length* as object properties, and extract these properties from standard part names.

The above-mentioned class *interference* represents an overlap relation between parts, possibly annotated with further data generated by the geometric analysis tool. The analysis tool generates instances of this class in an XML representation format, which the OntoDMU tool automatically converts into an ABox describing a collection of individuals inhabiting the class *Interference*, with additional data describing the participating parts and their classification according to the part ontology.

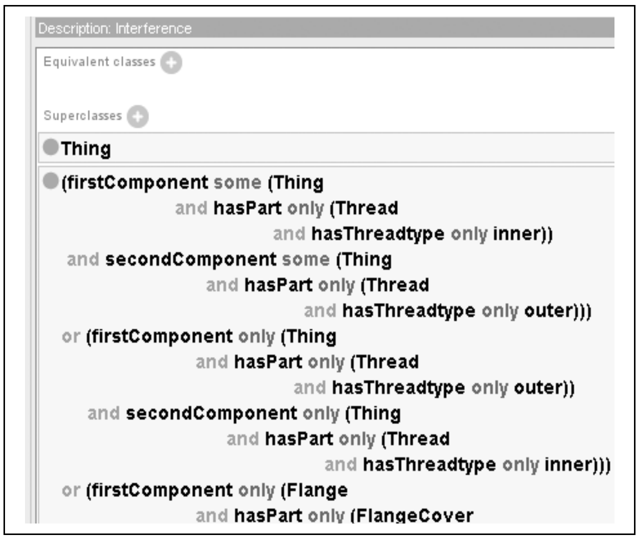
The classification of interferences as intended or faulty is now cast as a consistency check of the ABox thus generated with the background ontology, which must hence contain a formalization of rules stating what types of overlaps between parts are allowed, forbidden, or, in fact, mandatory. To see why these cases even arise, consider the following examples:

- Two bolts should never overlap; such and overlap, if detected, will always be classified as a design failure.
- A gasket, being deformable, may overlap with other parts. These, however, should be of a suitable type – e.g. a gasket should not overlap with a bolt, but may overlap with a flange
- Bolts in fact *must* always overlap with some other part (unless threads are explicitly modelled), namely with a part (typically a nut) having an inner thread, whose type and diameter match that of the bolt. These, however, are the *only* overlaps allowed for bolts.

It is an important design decision to which classes these pieces of knowledge should be attached in the background ontology. To balance the conflicting design goals of modularity, human readability, and efficiency of reasoning, we adopt the following approach. We attach general pieces of knowledge such as 'bolts should always (and only) overlap with parts that have inner threads' to the class *Interference* as a list of alternative exceptions.

Contrastingly, more specific information, such as that the part that a bolt overlaps with should have matching thread type and diameter, is attached to the relevant class of bolts, or more precisely to the relevant type of thread. E.g. the class representing the thread type M12 states that any part having an outer thread of this type may overlap only with parts having an inner thread of type M12.

Unfortunately, even for small part ontologies this leads to long and hard-to-parse lists of restrictions (Figure 6); we thus to some degree sacrifice human readability in favour of ease of machine processing: an alternative approach is to attach restrictions entirely to part classes instead of to the class *interference*. This leads to better modularity and is easier to read for humans. However, this requires an increased use of so-called inverse properties which traverse object properties *backwards* (in this case, from a part participating in a particular interference to the interference itself), which leads to increased processing time. In our experiments using Pellet, this meant an increase from about 1min 40s in the analysis of a particular CAD model to more than 15min.



```
Description: Interference
Equivalent classes

Superclasses
Thing
(firstComponent some (Thing
                and hasPart only (Thread
                          and hasThreadtype only inner))
  and secondComponent some (Thing
                  and hasPart only (Thread
                            and hasThreadtype only outer)))
  or (firstComponent only (Thing
                 and hasPart only (Thread
                           and hasThreadtype only outer))
    and secondComponent only (Thing
                    and hasPart only (Thread
                              and hasThreadtype only inner)))
  or (firstComponent only (Flange
                 and hasPart only (FlangeCover
```

Fig. 6 Additional restrictions on the concept *interference*.

Currently, restrictions relating standard and non-standard parts in the ontology are created manually. Our envisaged approach foresees to extract those relations from a PLM repository. A precondition for this extraction is to have annotated product structure items in the PLM/PDM system. Those annotations include a unique reference between a part and its concept.

5 A remark on using default logic

As discussed above, the *interference* concept captures most of the explicit allowed interferences. The effort for such an explicit interference modelling increases quadratically with the number of implemented concepts. An interesting point to be made is that for purposes of the specification of the background ontology, it would be useful to have a language with defaults available; see Chapter 6 of [29] for a brief explanation of defaults and an overview of existing approaches to adding them to ontology languages. Defaults are not currently included as a feature in OWL, which we continue to use nonetheless for sake of its well-developed reasoning support. We outline briefly how defaults could be employed to increase in particular the degree of modularity of the background ontology.

Roughly speaking, default implications state that given certain conditions, certain conclusions are expected to hold *normally*, but may be overridden when more specific information about the given situation becomes available. The standard example is that birds *normally* fly, thus leading us to conclude provisionally (*defeasibly*) that a particular bird flies unless more specific information about it becomes available, such as that the given bird is a penguin. In the concrete setting of specifying legal and illegal overlaps of parts in a CAD object, having such a mechanism available would simplify the overall structure of the ontology rather strongly: To begin, one would be able to just say that generally, parts should not overlap.

This general proviso would then be overridden by exceptions, such as that gaskets typically overlap with other specific parts having dedicated channels for the gasket; the latter could, again, be overridden in particular cases where the channel might be absent. This approach is *modular* because one can extend the range of available parts without having to adapt the general principles (e.g. we never have to adapt our initial default stating that parts do not overlap, even though exceptions to this keep accumulating).

Consequently, an OWL modelling of the situation has to circumvent this lack of expressivity rather visibly. E.g. all exceptions to the general rule that parts do not overlap are currently explicitly attached to the concept of an interference as shown above.

6 Conclusion

The fact that most current KBE solutions address only individual design problems and do not offer enterprise-level solutions largely goes back to problems of knowledge acquisition. Even though this level is typically covered by PLM systems and hence established information repositories do exist, there is to date no automated way for transforming this information into codified knowledge as it is typically used in knowledge based engineering rules.

To contribute to realising the vision of a sustainable capitalisation of the engineering information stored in PLM repositories as intellectual property assets, we have presented an approach to combining product structures and namespaces provided by PLM-systems on the one hand and significant domain specific standards on the other hand in order to establish a background ontology and thus create a powerful semantic representation of codified engineering knowledge.

Even though the current implementation of our OntoDMU tool, which translates the output of a standard geometric analysis tool into a knowledge representation format that allows for a connection to a background ontology of codified engineering knowledge, is still at the prototype stage, benefits to be gained by its usage already become clearly visible. The productive interplay between a practical design problem, which reappears in different design contexts, and the background ontology created as part of the framework, leads to a clear reduction of manual intervention in the validation of CAD objects.

The approach of having an ambient ontology of general engineering knowledge in parallel with an ontology of standard parts which is automatically generated from CAD part catalogues and PLM systems is promising and will be further elaborated. As a next step, an annotated product structure will be used in conjunction with the background ontology, thus allowing for semi-automatic generation and maintenance of ontologies also of non-standard parts.

At the same time, the work presented here constitutes an important prerequisite for ontological support in the actual core PLM processes: the semantically correct integration of data fed back to the manufacturer from the product during its life cycle requires a semanticized representation of design objects as facilitated by our ontological approach to CAD. Future steps in our research program include the implementation of a semantical underpinning of sensor data and other Middle-of-Life data to obtain intelligent decision support for the full product life cycle, thus enabling optimal feedback of PLM data into the design and development process. As demonstrated in the CAD case study presented here, we expect substantial added value from the use of fully automated ontological reasoning in modern DL engines (as opposed to a classical programming approach as pursued, e.g. in the ICAD system [30]) with respect to decision quality as well as extensibility and adaptability; here, the use of a standard ontology language such as OWL carries the promise of a high degree of both interoperability and sustainability. The approach via a core set of ontologies managed independently of the involved software tools both reduces the overall maintenance effort and enables an increased degree of knowledge reuse, with knowledge being made available to the entire range of CAx systems involved in the product life cycle.

References

1. Ameri F, Dutta D (2004) Product lifecycle management needs, concepts and components, Product Lifecycle Management Development Consortium, vol: PLMDC-TR3-2004.
2. Garcia P B, Fan I S (2008) Practitioner requirements for integrated Knowledge-Based Engineering in Product Lifecycle Management, International Journal of Product Lifecycle Management, vol: 3.
3. Enovia I (2009) ENOVIA - PLM Solutions - 3D Digital Collaboration - ENOVIA VPLM products - Dassault Systèmes, www.3ds.com/products/enovia/portfolio/enovia-v5/enovia-vplm/all-products/ Accessed 9 December 2009.
4. Fan I, Bermell-Garciá P (2008) International Standard Development for Knowledge Based Engineering Services for Product Lifecycle Management, Concurrent Engineering, vol: 16, 271-277.
5. Pugliese D, Colombo G, Spurio M (2007) About the integration between KBE and PLM, Advances in Life Cycle Engineering for Sustainable Manufacturing Businesses, 131-136.
6. OMG (2005) KBE Services for PLM - RFP.
7. Thomke S, Fujimoto T (2000) The effect of front-loading problem-solving on product development performance, Journal of Product Innovation Management, vol: 17, 128–142.
8. Skarka W (2007) Application of MOKA methodology in generative model creation using CATIA, Engineering Applications of Artificial Intelligence, vol: 20, 677-690.
9. Nacsa J, Bueno R, Alzaga A, Kovács G L (2005) Knowledge management support for machine tool designers using expert enablers., International Journal of Computer Integrated Manufacturing, vol: 18, 561-571.

10. Kulon J, Mynors D, Broomhead P (2006) A knowledge-based engineering design tool for metal forging, Journal of Materials Processing Technology, vol: 177, 331-335.
11. Danjou S, Lupa N, Koehler P (2008) Approach for Automated Product Modeling Using Knowledge-Based Design Features, Computer-Aided Design & Applications, vol: 5, 622–629.
12. Penoyer J A, Burnett G, Fawcett D J, Liou S Y (2000) Knowledge based product life cycle systems: principles of integration of KBE and C3P, Computer-Aided Design, vol: 32, 311-320.
13. Milton N (2008) Knowledge technologies, Monza: Polimetrica.
14. Prasad B (2005) What Distinguishes KBE from Automation, www.coe.org/coldfusion/newsnet/Jun-05/knowledge.cfm#1 Accessed 2 December 2009.
15. Sriram R D (2006) Artificial intelligence in engineering: Personal reflections, Advanced Engineering Informatics, vol: 20, 3-5.
16. Oldham K, Kneebone S, Callot M, Murton A, Brimble R (1998) MOKA - A Methodology and tools oriented to Knowledge-based engineering Applications, Changing the ways we work: shaping the ICT-solutions for the next century, Göteborg, 198.
17. Van der Velden C, Bil C, Yu X, Smith A (2007) An intelligent system for automatic layout routing in aerospace design, Innovations in Systems and Software Engineering, vol: 3, 117–128.
18. Colombo G, Mosca A, Sartori F (2007) Towards the design of intelligent CAD systems: An ontological approach, Advanced Engineering Informatics, vol: 21, 153-168.
19. Abad-Kelly J, Cebrián D P, Chulvi V (2008) An Ontology-based approach to integrating life cycle analysis and computer aided design.
20. IBM C (2009) IBM - CATIA Product Synthesis Discipline - All products in this discipline, www-01.ibm.com/software/applications/plm/catiav5/disciplines/prodsynth/products.html Accessed 8 December 2009.
21. Crnkovic I, Asklund U, Dahlqvist A P (2003) Implementing and integrating product data management and software configuration management, Artech House Publishers.
22. VDI 2230 (2003) Systematic calculation of high duty bolted joints Joints with one cylindrical bolt, VDI-Verlag, Düsseldorf.
23. Ansaldi S, Bragatto P, Camossi E, Giannini F, Monti M, Pittiglio P (2006) A knowledge-based tool for risk prevention on pressure equipments, Computer-Aided Design & Applications, vol: 3, 99–108.
24. W3C (2004) OWL Web Ontology Language Overview, www.w3.org/TR/2004/REC-owl-features-20040210/#s2 Accessed 10 December 2009.
25. Sirin E, Parsia B, Grau B C, Kalyanpur A, Katz Y (2006) Pellet: A practical OWL-DL reasoner, Journal of Web Semantics, vol: 5, 51-53.
26. Sattler U (2010) Description Logic Reasoners, www.cs.-manchester.ac.uk/~sattler/reasoners.html Accessed 10 December 2009.
27. W3C (2009) ManchesterSyntax - OWL, www.w3.org/2007/OWL/wiki/ManchesterSyntax Accessed 14 December 2009.
28. Mossakowski T, Maeder C, Lüttich K (2007) The Heterogeneous Tool Set, Tools and Algorithms for the Construction and Analysis of Systems, TACAS 07, O. Grumberg and M. Huth, eds., Springer, Heidelberg, 519-522.
29. Baader F, Calvanese D, McGuinness D L, Nardi D, Patel-Schneider P F (Eds.) (2003) The Description Logic Handbook, Cambridge University Press.
30. Bermell-Garcıá P, Fan I S (2002) A KBE System for the Design of Wind Tunnel Models Using Reusable Knowledge Components, International Congress on Project Engineering, Barcelona, 23-25.