# Towards Ontological Support for Principle Solutions in Mechanical Engineering

Thilo Breitsprecher
Dept. of Mech. Eng.
Friedrich-Alexander-Universität
Erlangen-Nürnberg

Mihai Codescu
Dept. of Comput. Sci.
Otto-von-Guericke-Universität
Magdeburg

Constantin Jucovschi
Michael Kohlhase
Comput. Sci.
Jacobs University Bremen

Lutz Schröder
Dept. of Comput. Sci.
FAU Erlangen-Nürnberg

Sandro Wartzack
Dept. of Mech. Eng.
FAU Erlangen-Nürnberg

## Abstract

The engineering design process follows a series of standardized stages of development, which have many aspects in common with software engineering. Among these stages, the principle solution can be regarded as an analogue of the design specification, fixing as it does the way the final product works. It is usually constructed as an abstract sketch (hand-drawn or constructed with a CAD system) where the functional parts of the product are identified, and geometric and topological constraints are formulated. Here, we outline a semantic approach where the principle solution is annotated with ontological assertions, thus making the intended requirements explicit and available for further machine processing; this includes the automated detection of design errors in the final CAD model, making additional use of a background ontology of engineering knowledge. We embed this approach into a document-oriented engineering design process, in which the background ontology and semantic annotations in the documents are exploited to trace parts and requirements through the design process and across different applications.

## 1 Introduction

Much like software engineering design (in an ideal world), design processes in mechanical engineering proceed in multiple stages successively refining abstract requirements into a final solution. This process of *systematic engineering design* is standardized in models that bear substantial resemblance to the V-model, such as the German VDI 2221 [?]. However, only the last stage in this process, corresponding to the actual implementation in software engineering, has well-developed tool support, in the shape of CAD systems that serve to document the final design. Other stages of the design process are typically documented in natural language, diagrams, or drawings. There is little or no support available for interconnecting the various stages of the design, let alone verifying that decisions made in one stage are actually implemented in the next stage.

In: Alexandre Rademaker and Vinay K. Chaudhri (eds.): Proceedings of the 6th Workshop on Formal Ontologies meet Industry, Rio de Janeiro, Brazil, 22-SEP-2014, published at http://ceur-ws.org

Here, we embark on a program to fill this gap, focusing for a start on the last step in the development process, in which we are given a *principle solution* and need to implement this solution in the final design, a CAD model. The principle solution fixes important design decisions in particular regarding physical layout, materials, and connections but does not normally carry a commitment to a fully concrete physical shape. It is typically represented by a comparatively simple drawing, produced using plain graphics programs (e.g. within standard presentation tools) or even by hand. As such, it has a number of interesting features regarding the way it does, and also does not, convey certain information. The basic issue is that while one does necessarily indicate only one concrete shape in the drawing, not all aspects and details of this sketch are actually meant to be reflected in the final design. Some of this is obvious; e.g. it is clear that slight crinkles in a hand drawing are not intended to become dents in the final product, and to some (possibly lesser) degree it is also clear that not everything that is represented as a straight line or a rectangle in a simple sketch will necessarily be realized by the same simple geometry in the actual design. Other aspects are less straightforward; e.g. symmetries in the drawing such as parallelism of lines or equal lengths of certain parts, right angles, and even the spatial arrangement and ordering of certain components may constitute integral parts of the principle solution or mere accidents of the sketch. Other aspects of the design may be indicated by standard graphical symbolism; e.g. crosses often represent bolts. To aid human understanding of the principle solution, it is typically accompanied by a natural-language explanation that (hopefully) clears up most of the ambiguities; other aspects of the design are understandable only in the context of sufficient implicit knowledge, i.e. based on the experience of the design engineer.

The approach we propose in order to strengthen and explicate the links between the stages of the design process is, then, to integrate the documents associated to each stage into a unified document-oriented engineering design process using a shared background ontology. This ontology should be strong enough to not only record mere hierarchical terminologies but also, in our concrete scenario of principle solutions, to capture as far as possible the qualitative design intentions reflected in the principle sketch as well as the requisite engineering knowledge necessary for its understanding. (It might be fruitful to combine this approach with work on sketch maps in GIS [?] in order to distinguish between intended and contingent parts of the sketch automatically.) Such an ontology will in particular support the tracing of concepts and requirements throughout the development process; we shall moreover demonstrate on an example how it enables actual *verification* of a final design against constraints indicated in the principle solution.

Technically, we realize this approach by means of a modular semantic middleware architecture, the *Multi-Application Semantic Alliance Framework* (MASally), which connects a system of knowledge management web services to standard applications – in particular document players and CAD systems – via a network of thin API handlers that essentially make the framework parametric in the choice of CAD system. Background knowledge and design intentions are represented in a modular ontology that provides material for user assistance and forms the basis for the verification of design constraints. The formalized engineering knowledge required for the latter task is managed within the heterogeneous logical framework provided by the *Heterogeneous tool set* HETS [?], with the *Web Ontology Language (OWL)* [?] playing the role of the primary representation logic for the sake of its good computational properties. Sources of ontological knowledge include, besides manually extracted knowledge on engineering and basic geometry, semantic annotations of the principle sketch and the extraction of assertional knowledge from a CAD model. We illustrate our framework by means of an example where we verify aspects of the design of an assembly crane against the principle solution.

## 2 A Document-Oriented Process with Background Knowledge

We recall the stages of the *engineering design process* according to VDI 2221 [?].

**S1 Problem**: a concise formulation of the purpose of the product to be designed.

**S2 Requirements List**: a list of explicitly named properties of the envisioned product. It is developed in cooperation between designer and client and corresponds to the user specification document in the V-model.

**S3 Functional Structure**: a document that identifies the functional components of the envisioned product and puts them into relation with each other.

**S4 Principle Solution**: an abstract sketch capturing the most important aspects of the design.

**S5 Embodiment Design**: a CAD design that specifies the geometry of the final product.

**S6 Documentation**: accompanies all steps of the design process.

An approach to vertical semantic integration of this process is outlined in [**?**]. In this paper we concentrate on step **S4**, as it offers the most obvious handles for adding value using semantic services, and also discuss in more detail the structure of the ontology that drives them.

### 2.1 Principle Solutions

According to Pahl and Beitz [**?**], one can develop a principle solution for a product by combining working principles that correspond to the sub-functions identified in the function structure of the product. The search for applicable working principles and their ensuing combination in the principle solution is essential for the product development. For example, the manufacturing costs are determined to a large extent by these decisions. However, a combination of working principles cannot be fully evaluated until it is turned into a suitable representation. At this highly creative stage of the design process, the engineer does not want to consider the formalities inherent to a full-fledged CAD system. For this reason, probably the most common representations of principle solutions in mechanical engineering are old-fashioned hand-drawn sketches. Developing the principle solution mainly involves the selection of materials, a rough dimensional layout, and other technological issues. The design engineer can refer to various support tools in the search for working principles, such as the design catalogues of Roth [**?**] and Koller [**?**]. The degree of detail of a sketch varies between the two main levels of the design: while at the assembly level, the focus is mainly on the topology of the product, to ensure compatibility of the principles to be combined, at the level of parts and sub-assemblies more attention is given to the actual shape of the product to be developed. In the following, we discuss an example of a representation of a principle solution.

### 2.2 Case Study: An Assembly Crane

Our main case study concerns an assembly crane for lifting heavy machine components in workshops. This example has been used in a practical design assignment for engineering students at FAU Erlangen-Nürnberg in the winter term of 2012. In this design exercise, students were given a principle solution (Figures 1 and 2) along with some requirements (e.g. specified maximum power consumption, maximum torque, and maximum weight) and were asked to design an embodiment. Thus we have realistic documents for phases **S4** and **S5** of a representative and non-trivial design task to study.

The assembly crane to be designed can be divided into modules performing various functions. The modules are indicated by numbers in the figure: the main frame with a vertical beam, a cantilever, and parallel horizontal base profiles (1); and a lifting system, consisting of an electrically powered winch unit (2), connected via a cable (3), which is guided via deflection rollers, to a crane hook (4). This allows lifting, lowering and holding the machine components to be assembled. The requirements of the crane, which have been defined in a previous step, concern the material to be used (standard steel profiles for high strength and stiffness), the topology (the legs of the crane must be parallel, the vertical and the horizontal cantilever are perpendicular, the motor (2) must not be attached to the frame within the crane's working space), dimensions (maximum total height, minimum space between base profiles, minimum cantilever length) and manufacturing process constraints (weldment of main frame profiles and bolt connection of winch unit and main frame).

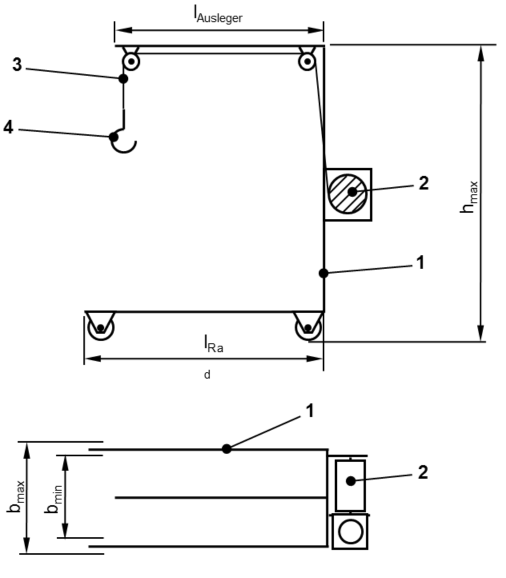


Figure 1: Principle Solution: Assembly Crane

Figure 2 details the principle solution of the winch unit: It consists of a drum (6a), which is welded (generally, the requirement of a weldment is indicated by a folded arrow) between two side plates (6b). In order to ensure correct reeling of the cable, the drum is thread-structured (6). The main shaft (7) is driven by an electric worm-geared flange motor (5) that is connected to the winch frame (11) via blind-hole bolts (indicated by crosses in the sketch). In order to decelerate the winch, to hold the load, and to allow emergency stops, a lamella disk break (9) is installed; it is connected to the main shaft by a suitable shaft-hub connection (10) that can withstand sudden increases in torque (e.g. due to emergency stops). An arrangement of locating and non-locating bearings (8) supports the main shaft. The ball bearings have to be arranged in such a way that axial forces are kept from the motor. The winch frame is realized as a stiff, yet weight-minimized, welded assembly, made of steel and is connected to the main frame of the crane with through-hole bolts.

# 3 Semantic Support for a Document-Oriented Engineering Design Process

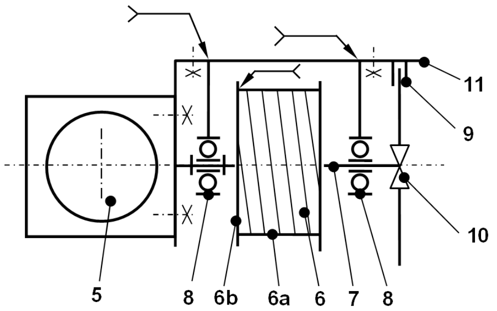


Figure 2: Principle Solution for the Winch Unit.

Every step of the engineering design process results in particular documents, e.g. text documents for **S1** to **S3** and **S6**, an image for **S4** (hand-drawn or produced in a simple graphics program), and a CAD assembly in **S5**. One of our goals is to integrate these into a document-oriented engineering design process, using semantic technologies.

## 3.1 A Semantic Annotation System

We build on the MASally architecture presented in [?] (under the name FFCad, considerably extended here to embrace document-oriented design processes), which assumes that the background knowledge shared by the manufacturer, design engineer, and the clients is reified into a flexiformal ontology (the cloud in Figure 3) and that the documents are linked into that ontology via a *semantic illustration mapping* (depicted by dashed arrows in Figure 3) from fragments or objects in the documents to concepts in the ontology (dotted circles), which may themselves be interconnected by ontology relations (solid arrows). The ontology itself is a federation of ontology modules describing different aspects of the engineering domain that are interconnected by meaning-preserving interpretations (see Section 4 for details). In the case of principle solutions, for instance, the semantic illustration mapping is currently generated from manual annotations of parts of the drawing with ontological concepts; to which degree annotations can be inferred by interpreting graphical jargon is the subject of further investigation.

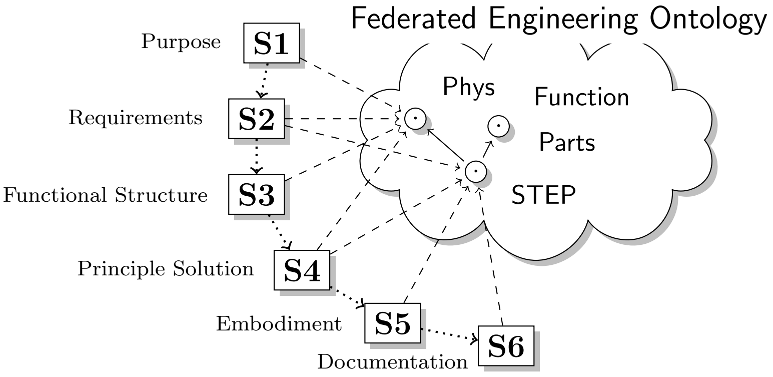


Figure 3: An Ontology-Supported Document-Oriented Design Process

In addition to the ontology links, we assume that the documents themselves are semantically linked via relations (the dotted arrows between the **S***i*) that model the process of goal refinement in the development process. These two primary relations are augmented with fine-grained annotations about document status, versioning, authorship, etc. Note that our approach crucially extends metadata-based approaches in that the annotations and relations point to document fragments – e.g. text fragments down to single symbols in formulae, regions in sketches, or shapes/sub-assemblies in CAD objects. All of these explicit annotations in the documents are the basis for semantic services that can be integrated into the documents (and their player applications) via the MASally framework, which we describe next.

## 3.2 Semantic Services via the MASally System

The Multi-Application Semantic Alliance Framework (MASally) is a semantic middleware that allows embedding semantic interactions into (semantically preloaded) documents. The aim of the system is to support the ever more complex workflows of product developers with their knowledge intensive tasks that so far only other humans have been able to perform without forcing them to leave their accustomed tool chain.

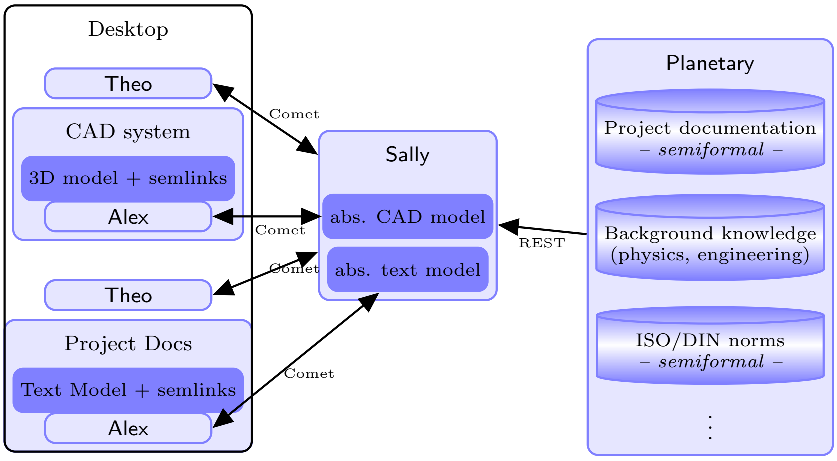


Figure 4: The MASally Architecture

The MASally system is realized as

- a set of semiformal knowledge management web services (comprised together with their knowledge sources under the heading Planetary on the right of Figure 4);
- a central interaction manager (Sally, the *semantic ally*) that coordinates the provisioning and choreographing of semantic services with the user actions in the various applications of her design process;
- and per application involved (we show a CAD system and a document viewer for **S4**/**S5** in Figure 4)
  - a thin API handler Alex that invades the application and relates its internal data model to the abstract, application-independent, content-oriented document model in Sally;
  - an instance of the application-independent display manager Theo, which super-imposes interaction and notification windows from Sally over the application window, creating the impression the semantic services come from the application itself.

This software and information architecture is engineered to share semantic technologies across as many applications as possible, minimizing the application-specific parts. The latter are encapsulated in the Alexes, which only have to relate user events to Sally, highlight fragments of semantic objects, handle the storage of semantic annotations in the documents, and export semantically relevant object properties to Sally. In particular, the Theos are completely system-independent. In our experience developing an Alex for an open-API application is a matter of less than a month for an experienced programmer; see [?] for details on the MASally architecture.

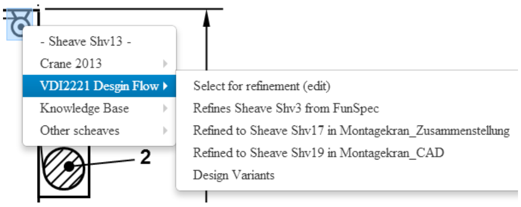
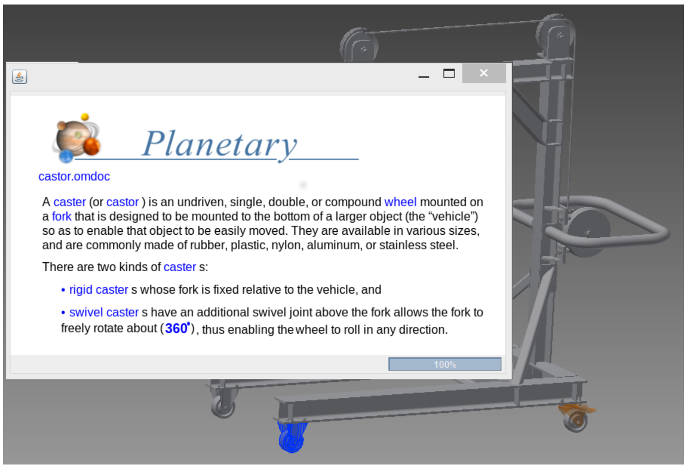

- Sheave Shv13 -
Crane 2013
VDI2221 Desgin Flow
Knowledge Base
Other scheaves
Select for refinement (edit)
Refines Sheave Shv3 from FunSpec
Refined to Sheave Shv17 in Montagekran_Zusammenstellung
Refined to Sheave Shv19 in Montagekran_CAD
Design Variants
2

Figure 5: Navigating the Refinement Relation

As a use case, let us consider the following situation. The design engineer is working on the principle solution from Figure 1 – a sketch realized as a vector image, displayed in an (in this case browser-based) image viewer. The user clicked on a detail of the sketch and received a (Theo-provided) menu that

1. identifies the object as 'Sheave S13' (the image is extended with an image map, which allows linking the region 'S13' with the concept of a 'sheave' in the ontology); further information about the object can be obtained by clicking on this menu item;
2. gives access to the project configuration that identifies the other documents in the current design;
3. gives access to the design refinement relation between the project documents: here, the object S13 is construed as a design refinement of the requirement S3 in the principle solution and has been further refined into objects S17 and S19 in the CAD assembly and the manufacturing plans generated from it;
4. allows direct interaction with the ontology (e.g. by definition lookup; see Figure 6, here triggered from the CAD system for variety);
5. gives shortcuts for navigation to the other sheaves in the current project.

Generally, the MASally system supports generic help system functionalities (definition lookup, exploration of the concept space, or semantic navigation: lookup of concrete CAD objects from explanations) and allows focus-



A caster (or castor ) is an undriven, single, double, or compound wheel mounted on a fork that is designed to be mounted to the bottom of a larger object (the “vehicle”) so as to enable that object to be easily moved. They are available in various sizes, and are commonly made of rubber, plastic, nylon, aluminum, or stainless steel.

There are two kinds of caster s:

- rigid caster s whose fork is fixed relative to the vehicle, and
- swivel caster s have an additional swivel joint above the fork allows the fork to freely rotate about (360°), thus enabling the wheel to roll in any direction.

Figure 6: Definition Lookup

preserving task switching (see [?] for a discussion). All we need for this are annotations of the VDI2221 relations, ontology links and of course the ontology itself, which we will discuss next.

## 4 The Federated Engineering Ontology

We proceed to discuss the design of the ontology that acts as the central repository of background knowledge. It serves as a synchronization point for semantic services, as a store for the properties of and relations between domain objects, and as a repository of help texts for the MASally system. As it has to cover quite disparate aspects of the respective engineering domain at different levels of formality, it is unrealistic to expect a homogeneous ontology in a single representation regime. Instead, we utilize the heterogeneous OMDoc/MMT framework [?, ?] that allows representing and interrelating ontology modules via meaning-preserving interpretations (i.e. theory morphisms). In particular, OMDoc/MMT supports the notion of meta-theories so that we can have ontology modules represented in OWL2 alongside modules written in first-order logic, as well as informal modules given in natural language. The OMDoc/MMT meta-morphisms relate all of these and moderate a joint frame of reference. Reasoning support is provided by the verification environment of the Heterogeneous Tool Set HETS [?], a proof management tool that interfaces state-of-the-art reasoners for logical languages. HETS mirrors the heterogeneity of the representation framework: new logics, logic translations or concrete syntaxes of languages can be plugged in without having to modify the heterogeneous and the deductive components of HETS. In our verification of design constraints, we employ, within OMDoc/MMT/HETS, the Distributed Ontology, Modeling and Specification Language DOL [?, ?] that provides specific support for heterogeneity in ontologies.

### 4.1 A Verification Methodology

We propose a general methodology for the verification of qualitative properties of CAD assemblies against principle solutions. While the checking of explicit *quantitative* constraints in principle solutions is supported by a number of research tools (e.g. the ProKon system [?]; in fact, some CAD systems themselves include constraint languages such as CATIA Knowledge Expert, which however are not typically interrelated with explicit principle solutions), there is to our knowledge currently no support for checking *qualitative* requirements given by the principle solution.

**Example 4.1.** In our case study introduced in Section 2.2, commonly encountered violations (in realizations produced by engineering students) of qualitative requirements in the principle solution were the following:

- the horizontal base profiles of the frame were not parallel;
- the types of weldments used did not ensure high stiffness and the local weldment area was not designed properly (e.g. missing ribs or stiffenings);
- the ball bearings were arranged in such a way that the non-locating bearing was closer to the motor, and thus the axial forces were transmitted into the motor.

We are going to use the requirement that the legs of the frame should be parallel as a running example throughout the rest of the section. It is clear that the other examples can be treated similarly.

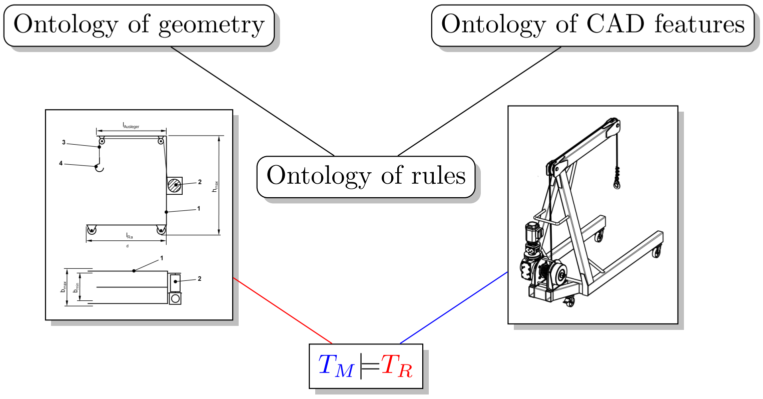


Figure 7: Verification of qualitative properties of CAD designs.

The first step is to provide a formal terminology for expressing the qualitative properties that a CAD design should fulfill. Since we are at the stage **S5** of the engineering design process, we have to collect requirements from all previous stages, in particular **S1** - explicit requirements - and **S4** - further restrictions on the acceptable designs introduced by the principle solution. Here, we concentrate on geometric properties of physical objects and therefore we tackle this goal by developing an ontology of geometric shapes. We then need to have means to formally describe the aspects of a CAD design that are relevant for the properties that we want to verify. Since we want to verify geometric properties, we are going to make use of an ontology of CAD features. We then need to formulate general rules regarding geometric properties of objects constructed by repeated applications of CAD features. This gives us a new ontology, of rules relating geometric properties and CAD features.

We now come to the task of verification of a concrete CAD design against the requirements captured by a given principle solution. In a first step, we generate a representation of the requirements as an ABox $T_R$ over the ontology of rules, in a way that will be explained below. The next step is to generate a representation of the CAD design as another ABox $T_M$ over the same ontology of rules, and then to make use of the rules to formally verify that $T_M$ logically implies $T_R$. This process is illustrated in Figure 7.

### 4.2 Ontology of Shapes

We begin setting up our verification framework by developing an ontology of abstract geometric objects, with their shapes and properties. The shape of a geometric object would seem to be a well-understood concept; however, the task of formalizing the semantics of shapes and reasoning about them is difficult to achieve in a comprehensive way. For a broader discussion, including some attempts to develop ontologies of geometric shapes, see, e.g., the proceedings of the Shapes workshop [**?**].

Our ontology, inspired by CYC [**?**], concentrates on geometric primitives of interest for CAD design . The central concept is that of *PhysicalObject*, which may be of an unspecified shape or can have a 2-dimensional or 3-dimensional shape. The object and data properties of the ontology are either parameters of the geometric shapes (e.g. diameter of a circle, or length of the sides of a square) or general geometric properties, like symmetric 2D- and 3D-objects and parallel lines.

**Example 4.2.** We present the fragment of the ontology[1] of shapes that is relevant for asserting that two objects are parallel, a DOL specification that extends our OWL formalization of geometry with the axiom that two coplanar lines are parallel if the angles of their intersections with a third line are equal. Since the intersection of two lines is a three-place relation, the two intersecting lines and the angle between them, we use reification to represent it as a concept *Intersection*, together with a role *intersectsWith* that links to the first constituent line, a class *LineAngle* for pairs of lines with angles (with associated projection roles) and a role *hasLineAngle* that links to the pair of the second line of an intersection and the angle between the two lines:

[1] The ontology modules relevant for the running example are available at `http://ontohub.org/fois-ontology-competition/FormalCAD/`.

$$\begin{tikzcd}
I_1 \arrow[r, "\mathit{hasLineAngle}"] \arrow[d, "\mathit{intersectsWith}"'] & (c, \alpha) & I_2 \arrow[l, "\mathit{hasLineAngle}"'] \arrow[d, "\mathit{intersectsWith}"] \\
\mathrm{a} \arrow[rr, "\mathit{isParallelWith}"'] & & \mathrm{b}
\end{tikzcd}$$

We denote the inverses of *hasLineAngle* and *intersectsWith* with *lineAngleOf* and *hasIntersection*, respectively. Since OWL does not support intersection of roles, the property will be expressed as a SWRL rule, that we write here informally to save space:

```
  isCoplanar(?a, ?b) ∧ hasIntersection(?a, ?i1) ∧ hasLineAngle(?i1, ?la) ∧ lineAngleOf(?la,
?i2) ∧ intersectsWith(?i2, ?b) ⇒ isParallelWith(?a, ?b)
```

### 4.3 Ontology of CAD Features

Inspired by [?], our ontology of features contains information about the geometry and topology of CAD parts. It describes assemblies and their parts, feature constructors and transformers, 2D sketches and their primitives, and constraints (cf. Example 4.3).

**Example 4.3.** We present a fragment of the ontology of features that is relevant for verifying that two objects are parallel. We have a concept of *3DPart* of an assembly and each part has been constructed in a 3D space which has 3 axes of reference. We record this by an object property *hasAxis*, with the inverse *isAxisOf*. Furthermore, 3D parts can be *constrained* at the assembly level. The constraint of interest for us is an angle constraint that specifies the angle formed between two axes, two edges or two faces of two chosen parts. Since this is again a relation with three arguments, we use reification, in a similar way as in Example 4.2, that is, we have a class *AngleConstraint* and three roles, *firstConstrainedLine* and *secondConstrainedLine* giving the two lines that are constrained and *constrainedAngle* giving the specified angle.

In a similar way, a *Mate* constraint determines that the two axes are in the same plane.

### 4.4 Ontology of rules

The next step is to relate via rules the concrete designs using feature transformers and constructors, given as elements of the ontology of features, to the abstract shapes in the ontology of geometry. It is worth mentioning that the rules can be themselves subject to verification (a proof of concept was given in [?]). The advantage of our approach is that the task of verifying the rules, which can be quite complex, is separated from the task of checking correctness of individual CAD designs, which makes use of the rules.

**Example 4.4.** The DOL *alignment* below states that each part is a physical object and that lines and angles in the same ontologies are equivalent (we elide namespaces).

**alignment** Base : Features **to** Shapes =
*Line* = *Line*, *Angle* = *Angle*, *3DPart* < *PhysicalObject*

The outcome is that we can use DOL *combinations* to put together the two ontologies while taking into account the semantic relations given by the alignment. We further state that an angle constraint in an assembly gives rise to an intersection between the constrained lines and that two parts of an assembly are parallel if their axes are parallel. We use Manchester syntax for OWL, with **o** denoting role composition.

**ontology** Rules = **combine** Base **then**
**ObjectProperty:** *intersOfAngleConstraint*
**Domain:** *AngleConstraint* **Range:** *Intersection*
**ObjectProperty:** *isParallelWith*
**SubPropertyChain:** *hasAxis* **o** *isParallelWith* **o** *isAxisOf*
**ObjectProperty:** *firstConstrainedLine*
**SubPropertyChain:** *intersOfAngleConstraint* **o** *intersectsWith*
**ObjectProperty:** *secondConstrainedLine*
**SubPropertyChain:** *intersOfAngleConstraint* **o** *hasLineAngle* **o** *lineOfLineAngle*
**ObjectProperty:** *constrainedAngle*
**SubPropertyChain:** *intersOfAngleConstraint* **o** *hasLineAngle* **o** *angleOfLineAngle*

Similarly, we have a rule saying that two objects that are in planes constrained using *Mate* are coplanar.

### 4.5 Generating the ABoxes and proving correctness

The principle solution is available as an image file, together with a text document that records additional requirements introduced in the principle solution, thus further restricting the acceptable realizations of the design. Each part of the sketch has been identified as a functional part of the principle solution and given a name; this yields the required individual names for our ABox. The assertions regarding the individuals thus obtained are added as semantic annotations to the text that accompanies the image (Figure 8).

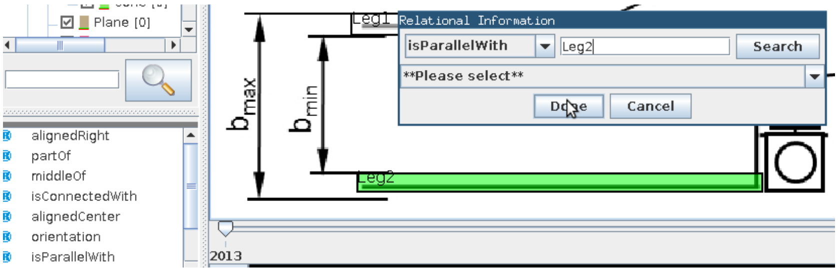


Figure 8: Making assertions regarding individuals explicit using AKTiveMedia [?]

**Example 4.5.** The following ABox expresses that the parts identified as *leg1* and *leg2* of the principle solution should be parallel:

**ontology** PRINCIPLESOLUTION = RULES **then**
  **Individual:** *Leg2* **Individual:** *Leg1* **Facts:** *isParallelWith Leg2*
. . .

The ABox of the CAD design is generated from its history of construction, using the Alex for CAD. The following part of this ABox expresses that the two legs of the crane have been explicitly constrained to be perpendicular to the main frame and coplanar in the CAD model:

**ontology** CADMODEL = RULES **then**
**Individual:** *a1* **Types:** *Line* **Individual:** *a2* **Types:** *Line* **Individual:** *a3* **Types:** *Line*
**Individual:** *craneLeg1* **Types:** *3DPart* **Facts:** *hasAxis a1*
**Individual:** *craneLeg2* **Types:** *3DPart* **Facts:** *hasAxis a2*
**Individual:** *frameBase* **Types:** *3DPart* **Facts:** *hasAxis a3*
**Individual:** *alpha* **Types:** *Angle* **Facts:** *valueOf 90*
**Individual:** *ac1* **Types:** *AngleConstraint*
**Facts:** *firstConstrainedLine a1*, *secondConstrainedLine a3*,*constrainedAngle alpha*
**Individual:** *ac2* **Types:** *AngleConstraint*
**Facts:** *firstConstrainedLine a2*, *secondConstrainedLine a3*, *constrainedAngle alpha*
. . .

Following Figure 7, we have to show that all models of the ABox generated from the CAD design are models of the ABox generated from the principle solution. DOL uses *interpretations* to express this; e.g., checking that the two legs of the crane are parallel amounts to checking correctness of the DOL interpretation

**interpretation** VERIF :PRINCIPLESOLUTION **to** CADMODEL =
  *Leg1* $\mapsto$ *craneLeg1*, *Leg2* $\mapsto$ *craneLeg2*

using one of the provers interfaced by HETS, e.g. the Pellet reasoner for OWL [?]; as expected, the reasoner makes short work of this.

## 5 Related Work

In previous work [?], we have developed an export function from SOLIDWORKS that generates from the internal representation of a CAD design a description of its construction in a variant of higher-order logic. One can then relate this construction to abstract geometric shapes and prove this relation to be correct using a higher-order proof assistant. In the context of the methodology introduced in Section 4.1, each such relation between the construction and its abstract geometric counterpart gives rise to a formally verified rule in the ontology. At the informal level, we have moreover developed a semantic help system for CAD objects based on the Semantic Alliance Framework [?], and have illustrated the use of this information for semantically supported task switching [?]. Our methods should be seen as distinct from the use of ontologies in workflow management and web service composition, as, e.g., in the Taverna system [?], in that we aim to semanticize the actual content of documents.

Several ontologies of features have been developed, with the typical application scenario being interoperability and data interchange between CAD systems, rather than verification of qualitative properties of CAD assemblies. E.g., OntoSTEP [?] enriches the semantics of CAD objects represented in the system-independent ISO-standard interchange format STEP. Our heterogeneous approach allows integrating OntoSTEP (or any other ontology of features) into our federated engineering ontology and relating it to our ontology of features, without having to modify our verification methodology.

Various approaches have been explored to integrate semantics into the engineering design process. E.g., *features* as used in *feature technology* [?] aggregate geometric objects and semantics. Different types of features are defined (eg. form features, semantic features, application features, compound features), depending strongly on the technical domain and the product life-cycle phase in which features are used. We expect features to play a role in further semanticizing step S5 (embodiment, Section 2) in future work.

As mentioned in Section 2.1, the most common representations of principle solutions in mechanical engineering are probably hand-drawn sketches. One alternative approach is the Contact-and-Channel Model (CCM) [?]. The basic idea is that every technical system can be represented as a system of *working surface pairs* and *channel and support structures*. In our case study, an example of a working surface pair would be the shaft-hub connection between the winch main shaft and the lamella disk break, and the main shaft, where the break torque of the disk break is *channeled* to the winch drum in order to stop the cable, would be a channel and support structure. Approaches of this kind, in combination with ontological models of function (e.g. [?]; see also the survey by Erden et al. [?]), are candidates for integration with our ontological process model in future extensions covering the step from the function structure to the principle solution.

## 6 Conclusions

We have described a framework for semantic support in engineering design processes, focusing on the step from the principle solution to the embodiment, i.e. the CAD model. We base our framework on a flexiformal background ontology that combines informal and semiformal parts serving informational purposes with formalized qualitative engineering knowledge and formal semantic annotation of principle sketches. The latter serve to separate contingencies of the sketch from its intended information content, and enable *automated* verification of the CAD model against aspects of the principle solution. We combine this approach with a document-oriented process that relies on the background ontology for tracking the identity of parts through the design process and across different applications, which are accessed in a unified manner within the MASally framework.

As a proof of concept, we have illustrated our approach on the partial verification of a CAD model of an assembly crane against the principle solution, showing in particular that the ability to draw logical inferences is important when verifying qualitative constraints. This allowed the system to, e.g., accept two parts as satisfying a parallelism constraint formulated in the principle solution although the CAD model did not mention such a constraint, which instead had to be inferred from other constraints in the model.

We currently use OWL as the logical core of our verification framework, representing the requisite background knowledge in a TBox and generating ABoxes from the principle sketch and the CAD model. Our approach is based on heterogeneous principles, through use of the Heterogeneous Tool Set Hets and the Distributed Ontology, Modeling and Specification Language DOL [?, ?]. It is thus easily possible to go beyond the expressivity boundaries of OWL where necessary, e.g. by moving some parts of (!) the ontology into first-order logic or, more conservatively, by using rule-based extensions of OWL such as SWRL [?] — this will increase the complexity of reasoning but the Hets system will localize this effect to those parts of the ontology that actually need the higher expressive power. Use of SWRL will in particular increase the capabilities of the system w.r.t. arithmetic reasoning.

## References

[1] Albert Albers and Christian Zingel. Extending SysML for engineering designers by integration of the contact and channel–approach (CCM) for function-based modeling of technical systems. In *Systems Engineering Research, CSER 2013*, volume 16 of *Proc. Comput. Sci.*, pages 353 – 362. Elsevier, 2013.

[2] Raphael Barbau, Sylvere Krima, Rachuri Sudarsan, Anantha Narayanan, Xenia Fiorentini, Sebti Foufou, and Ram D. Sriram. OntoSTEP: Enriching product model data using ontologies. *Computer-Aided Design*, 44:575–590, 2012.

[3] Stefano Borgo, Massimiliano Carrara, Pawel Garbacz, and Pieter Vermaas. A formal ontological perspective on the behaviors and functions of technical artifacts. *AI EDAM*, 23:3–21, 2009.

[4] Thilo Breitsprecher, Mihai Codescu, Constantin Jucovschi, Michael Kohlhase, Lutz Schröder, and Sandro Wartzack. Semantic support for engineering design processes. In *Int. Design Conf., DESIGN 2014*, pages 1723–1732. Design Society, 2014.

[5] Gino Brunetti and Stephan Grimm. Feature ontologies for the explicit representation of shape semantics. *J. Comput. Appl. Technology*, 23:192–202, 2005.

[6] Ajay Chakravarthy, Vitaveska Lanfranchi, and Fabio Ciravegna. Requirements for multimedia document enrichment. In *World Wide Web, WWW 2006*, pages 903–904. ACM, 2006.

[7] Catalin David, Constantin Jucovschi, Andrea Kohlhase, and Michael Kohlhase. Semantic Alliance: A framework for semantic allies. In Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, editors, *Intelligent Computer Mathematics, CICM 2012*, volume 7362 of *LNAI*, pages 49–64. Springer, 2012.

[8] M. Erden, Hitoshi Komoto, Thom van Beek, Valentina D'Amelio, E. Echavarria, and Tetsuo Tomiyama. A review of function modeling: Approaches and applications. *AI EDAM*, 22:147–169, 2008.

[9] Ian Horrocks, Peter Patel-Schneider, Sean Bechhofer, and Dmitry Tsarkov. OWL rules: A proposal and prototype implementation. *J. Web Sem.*, 3:23–40, 2005.

[10] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a web ontology language. *J. Web Semantics*, 1:7–26, 2003.

[11] Sahib Jan, Angela Schwering, Malumbo Chipofya, and Jia Wang. Qualitative representations of schematized and distorted street segments in sketch maps. In *Spatial Cognition 2014*, LNCS. Springer, 2014. To appear.

[12] Andrea Kohlhase, Michael Kohlhase, Constantin Jucovschi, and Alexandru Toader. Full semantic transparency: Overcoming boundaries of applications. In Paula Kotzé, Gary Marsden, Gitte Lindgaard, Janet Wesson, and Marco Winckler, editors, *Human-Computer Interaction, INTERACT 2013*, volume 8119 of *LNCS*, pages 406–423. Springer, 2013.

[13] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*, volume 4180 of *LNAI*. Springer, 2006.

[14] Michael Kohlhase. Knowledge management for systematic engineering design in CAD systems. In Franz Lehner, Nadine Amende, and Nora Fteimi, editors, *Professionelles Wissenmanagement, ProWM 2013*, pages 202–217. GITO, 2013.

[15] Michael Kohlhase, Johannes Lemburg, Lutz Schröder, and Ewaryst Schulz. Formal management of CAD/-CAM processes. In Ana Cavalcanti and Dennis Dams, editors, *Formal Methods, FM 2009*, volume 5850 of *LNCS*, pages 223–238. Springer, 2009.

[16] Rudolf Koller and Norbert Kastrup. *Prinziplösungen zur Konstruktion technischer Produkte*. Springer, 1994.

[17] Martin Kratzer, Michael Rauscher, H Binz, and P Göhner. Konzept eines Wissensintegrationssystems zur benutzerfreundlichen, benutzerspezifischen und selbständigen Integration von Konstruktionswissen. In *Design for X, DFX 2011*. TuTech Innovation, 2011.

[18] Oliver Kutz, Mehul Bhatt, Stefano Borgo, and Paulo Santos, editors. *The Shape of Things, SHAPES 2013*, volume 1007 of *CEUR Workshop Proc.*, 2013.

[19] D. Lenat. Cyc: A Large-Scale Investment in Knowledge Infrastructure. *CACM*, 38:33–38, 1995.

[20] Till Mossakowski, Oliver Kutz, Mihai Codescu, and Christoph Lange. The distributed ontology, modeling and specification language. In *Modular Ontologies, WoMo 2013*, volume 1081 of *CEUR Workshop Proc.*, 2013.

[21] Till Mossakowski, Christoph Lange, and Oliver Kutz. Three semantics for the core of the distributed ontology language (extended abstract). In Francesca Rossi, editor, *International Joint Conference on Artificial Intelligence, IJCAI 2013.* IJCAI/AAAI, 2013.

[22] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set, Hets. In *Tools Alg. Constr. Anal. Systems, TACAS 2007*, volume 4424 of *LNCS*, pages 519–522. Springer, 2007.

[23] G Pahl, W Beitz, J Feldhusen, and K.-H. Grote. *Engineering Design.* Springer, 3rd edition, 2007.

[24] Florian Rabe and Michael Kohlhase. A scalable module system. *Inf. Comput.*, 230:1–54, 2013.

[25] Karlheinz Roth. *Konstruieren mit Konstruktionskatalogen.* Springer, 1994.

[26] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *J. Web Semantics*, 5:51–53, 2007.

[27] VDI. *Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte (Systematic approach to the development and design of technical systems and products) – VDI 2221*, 1993.

[28] VDI. *Informationsverarbeitung in der Produktentwicklung – Feature-Technologie (Information technology in product development – Feature Technology) – VDI 2218*, 2003.

[29] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria Balcazar Vargas, Shoaib Sufi, and Carole Goble. The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucl. Acids Res.*, 2013.