From form features to semantic features in existing MCAD: an ontological approach

From form features to semantic features in existing MCAD: an ontological approach

F. Mandorli[a] and S. Borgo[b]* and P. Wiejak[a,b]

[a]*Department of Industrial Engineering and Mathematical Sciences, Polytechnic University of Marche, Ancona, Italy;*
[b]*Laboratory for Applied Ontology, ISTC CNR, Trento, Italy;*

*Stefano Borgo
Via alla cascata 56C, 38123 Povo Trento, Italy
stefano.borgo@cnr.it

Abstract: Today's product development processes rely on Mechanical Computer-Aided Design (MCAD) systems that implement a geometric-centered perspective in design. The development of long discussed feature-based MCAD has not yet led to systems that truly support semantic and functional representation of features, which hampers also the use of these models for functional reasoning. This paper investigates the present feature-based MCAD limitations and illustrates, through simple examples, how ontological analysis and feature re-classification can drive the introduction of software extensions to achieve in existing MCAD a first level of semantic representation of features enhancing the cognitive transparency of the final model. The proposal also shows how to automatically validate these features from the functional viewpoint.

Keywords: feature; MCAD; ontological analysis; semantic representation; design rational; functional feature.

Introduction

Mechanical Computer-Aided Design (MCAD) systems play an important role in today's product development processes: they are used to define the digital models required to perform virtual tests and analysis (to have feedback during the design development process), as well as to support technical communication, by means of representing shape related information, like geometry and dimensions. However, these MCAD systems are still unsatisfactory on important aspects, and this causes difficulties in changing and re-

using previously developed models. This is a severe limitation if we consider the intrinsic nature of the design and product development processes that, being highly dynamic, require frequent model alteration.

In particular, present MCAD systems have been developed taking a geometric-centered perspective and lack support for the design rationale as well as for semantic and functional representation of elements of the model (Otto et al. 2015). Note our use of the terminology: we use *design rationale* to refer to the motivations for the design model (the assumptions behind it, the solution principle, the working structure, the chosen layout etc.) We write *semantic feature* to mean the ontological and the cognitive meanings of a design element. Here, 'ontological' refers to the way features are understood and to the essential relationships they hold with the rest of the model; cognitive refers to the way experts understand the contribution of the feature to the properties and capabilities of the overall model and its physical realizations. Finally, we call *functional feature* the teleological view of a feature in the design, what is usually known as its functionality or purpose. We remind the reader that, since functionality is part of the ontological modeling, for each functional feature there is a corresponding semantic feature. In today's MCAD the user can handle geometric information in different ways, but she cannot directly relate, manage or represent the design rationale that is behind the decision to model a certain shape. Similarly, neither the ontological and cognitive meanings of a feature (semantic feature) nor the specific teleological view (functional feature) can be expressed in the MCAD model.

While the authors agree with these observations, they challenge the overall conclusion. The objective of the paper is to investigate the present feature-based MCAD limitations from an ontological perspective and to illustrate, through simple examples, how ontological analysis and re-classification of features can lead to the implementation of tools that support semantic, and in particular functional, representations of features. Practically, the paper provides answers to the following research questions:

- RQ1: How can one overcome the gap between functional and geometrical features in today's MCAD systems?
- RQ2: Can one introduce verification checks in existing MCAD systems to ensure that a semantic feature, introduced during the development of the model, preserves its functionality in the final model, and how?

In our view, a *truly* feature-based MCAD requires the design of a new generation software core that works explicitly with semantic features as opposed to sets of data and parameters. Since we still lack an understanding of features suitable to guide the building of such a core and in the hope to pave the way for the development of new approaches, this paper investigates partial, yet practical and effective, solutions that allows to identify semantic feature types manageable in existing off-the-shelf MCAD systems. This proposal cannot possibly cover all semantic features, but has the advantage of relying on the information that modern MCAD systems actually manage.

Structure of the paper: The next section introduces today's MCAD systems and their approach on feature modeling. The following section focuses more broadly on engineering features and basic ontological distinctions. Next, the core of existing MCAD feature commands is analyzed and its limitations discussed. Then, our novel approach is introduced including a reclassification of hole and slot features based on their semantics and functionality. A discussion of what has been achieved concludes the paper.

MCAD models and MCAD features

Most of today's commercially available MCAD systems adopt the so-called feature-based modeling approach. This design-by-feature paradigm assumes that a mechanical part can be described by a set of features, which captures and represents the design intent of the part. While a review of the literature on feature technology and design intent representation is beyond the scope of this paper, overviews of the origin of the feature-based technology from the early 1980s already exist, e.g., (Shah and Mantyla 1995), as well as more recent reviews of the concept of feature from the ontological viewpoint (Sanfilippo and Borgo 2016; Li et al. 2020) and surveys about the design intent concept (Otey et al. 2018).

Feature-based MCAD systems have been introduced in the late 1990s. The main intent was to provide the users with a more efficient and user-friendly modeling environment. Feature-based systems introduced features as modules (sets of interrelated geometrical information) in the existing modeling techniques, based on the explicit use of Boolean operations, and added parametrization. In a parametric feature-based model dimensional information is locally stored within the feature module and the list of features is stored following the order of their introduction in the model, the so-called *feature history.* By selecting a feature from the list, the user can access the corresponding set of information

and change the feature dimensions in terms of parameters and generate modified versions of the feature which is then cast into the model.

This implementation of features is quite trivial as it takes features to be geometrical entities, indeed they are just clusters of geometrical data. From now on we call these *MCAD features*. More precisely, MCAD features are features for which today's MCAD systems have commands available. (Appendix A shows the classification of the most common MCAD-feature commands.) The semantic feature, which includes the ontological and cognitive levels and, in turn, also the functional level, are ignored. A direct consequence of such situation is that MCAD novices often end up with feature-based models comprising very complicated (and sometimes ill-defined) sequences of features. Even expert users can find problematic to keep control of the status of and interactions across the different MCAD features when changes are introduced in the model. Since MCAD models tend to include many intertwined features, a MCAD user refrains from altering a model created by others because the consequences in the model of changes in a MCAD feature can be unpredictable. Semantically speaking, this problem is due to the fact that the logical connections between the generated shape and the rationale for the features' introduction is missing in the model.

The lack of semantic information in CAD models causes a transparency problem and determines the unsatisfactory way in which MCAD is used today. Indeed, in MCAD practice and industrial applications users often prefer to re-model from scratch than to modify an existing complex model. The latter activity, which in principle is the natural choice, is considered too error-prone, and it is somehow paradoxical that advanced CAD technology produces models that in practice are considered unchangeable obstructing the natural desire for efficient model re-use.

While academic research has been focused on defining suitable methodologies to limit the risk of creating incorrect feature-based models (Bodein, Rose, and Caillaud, 2014; Camba, Contero, and Company, 2016; Mandorli and Otto, 2013; Cheng and Ma, 2017), MCAD vendors have tried to cope with the transparency problem by adding new modeling commands to existing MCAD systems. These commands aim to overcome the problem of altering complex feature-based models by enabling direct modifications of the model shape via stretching, turning and twisting the geometric entities of the boundary model representation (B-rep) (Otto et al. 2015). This approach, called *direct modeling* or *explicit modeling* in the context of commercially available MCAD systems, might be capable of solving some specific problems, especially related to the alteration of imported

models that have lost the feature history structure. However, from the methodological point of view, direct modeling does not address the problem of model reuse and does not tackle its cause: the lack of semantic information in MCAD models. To appreciate this point, in the next section we take a broader perspective and look at the aims and expectations of semantic feature modeling.

MCAD models and product features: a broader perspective

In the ideal view of the product lifecycle the information flow starts from the collection of data about case scenarios, user requirements and standards' constraints, and moves to the characterization of possible functional and structural solutions as well as to the evaluation of how much these solutions match the desired results within business and market conditions. This view is clearly a simplification since, for instance, information about later stages of actual product lifecycles has to be taken into account in the early phases of the process. In the analysis of both the theoretical and the actual information flows the information gathered to design a specific solution is quite heterogeneous as it covers data about physical components, materials, object behaviors and so on.

MCAD systems have been developed to mainly support the 'embodiment', and 'detail' design phases (Pahl et al. 1996). At this stage, the generic concept of function has been subdivided into sub-function and sub-assembly, up to the point of assembly components and elementary functions. These can be directly related to specific form features, that are part of the overall shape of the component (e.g. shape of components aimed to transmit torque, to define threaded or elastic fasteners, to create linear or rotational guide, etc.).

We have seen in the previous section that today these aspects are primarily modelled in terms of features to the point that the product itself is essentially seen as a bundle of features (Shah and Mantyla 1995). From an ontological viewpoint, one distinguishes two types of feature, physical features and information features (Sanfilippo and Borgo 2016). A physical feature, hereafter *p-feature*, is "a physical entity that makes up some physical part" of a product item like a hole, a protrusion, a surface smoothness on a mechanical component. A p-feature is a part or a quality of the manufactured physical object. Physical features have their counterpart at the design level, the so-called information features or *i-features*. Information features are elements not of a product item but of the product description. They are modeling entities "that allow commonly used shapes to be characterized [...] with a set of attributes relevant to an application" (Shah and Mantyla

1995). Information features exist only as parts of a description of the product, e.g., the hole or of the slot in the MCAD model of a mechanical component.

A p-feature of a product item (e.g., the hole of a manufactured component) is said to realize the i-feature of its MCAD model (the design of the hole of the component) when the manufactured product (the mechanical component) is created in a manufacturing process with the aim to realize that MCAD model (the mechanical component design). The distinction between p-feature and i-feature is sharp in ontological terms and the dependence between these two notions is clearly strong from both the ontological and the engineering point of view. Indeed, the rationale of an i-feature in a MCAD design is related to the designer's expectations on the realizations of that MCAD design, i.e., any product manufactured according to that design. In this sense, the rationale of an i-feature can be tracked down to the need of the corresponding p-feature in the product item.[1]

This ontological dependence between p-features and i-features is what ensures the correspondence between the functional realization performed by the physical object (the product) and the functional representation and reasoning based on the information object (the design). Functional information has been modeled from different perspectives in engineering design and ontology, see e.g. (Chandrasekaran 2005, Hirtz et al. 2003, Kitamura et al. 2006, Garbacz et al. 2011). In dealing with MCAD modeling, it seems more appropriate to focus on the behavioral view of function according to which a product has a specific function if it can manifest a certain desired behavior in a suitable scenario (Chandrasekaran 2005). Since playing a role is to manifest a given behavior, an ontologist would translate the expression 'product X has function F' into 'product X is a role-holder (i.e. a possible player) to F' (Mizoguchi, Kitamura, and Borgo 2016). The need to manifest a behavior directly motivates most of the p-features that characterize a manufactured product (not all since some features have a social connotation, e.g. serve to identify the object as belonging to a certain brand, like a logo, or to a special type, like an emergency switch). The p-features of a product may cover a large spectrum of types: from the realization of geometric shapes to the quantity of used material, from the capacity to resist deformation to the manifestation of ergonomic qualities.

Given this general view, what can we say from the ontological viewpoint of MCAD features, that is, i-features as modeled by existing MCAD ? Due to the nature of today's

[1] Some features may be introduced only for making possible a manufacturing process. Here we treat them as any other feature type.

MCAD systems, MCAD features are structurally grounded in the geometric viewpoint underlying the MCAD engine. This means that while their rationale stems from semantic and functional analysis, their representation is purely geometrical. The geometrical implementation even of a functional characterization is not unique. A join function applied to two components can be realized by using screws, which need a through hole on one piece and a blind threaded hole in the other, or by using screws and nuts, which need a through hole in both pieces. A through hole feature can be motivated by a join function as well as by a convey function, thus there is no one-to-one relationship between functional and geometrical characterization of the feature. MCAD features rely on the geometrical characterization only, while for model reuse one would need to have available the i-feature rationale, or at least its functional view. This gap hinders the reuse of today MCAD models and motivates the first research question that we raised in this paper: can one overcome the gap between functional and geometrical features in MCAD systems as we know them today?

One way to overcome the problem is to analyze the geometrical parameters upon which MCAD features rely, and to ask if these parameters can be somehow differentiated and rearranged by relevance. Is the depth value of this hole as important as the position of its axis? Is the value of the diameter of this hole as important as the position of its bottom face? When the answer is positive, that is, when parameters are equally relevant, we can ask what this information is telling us about the rationale of that feature. When the answer is negative, that is, when parameters are not equally relevant, we may try to introduce an ordering on the parameters to encode functional information.

This encoding of the functional rationale of i-features, if it works, answers only part of the problem. A basic requirement to reuse MCAD models is the possibility to verify that the introduction in a model of a MCAD feature, or the change of an existing one, does not affect the properties that characterize the MCAD features introduced earlier. A product model should ensure that any realization of the model manifests the expected behaviors for which the design was conceived. The correct presence in the model of all the expected i-features is a precondition to this goal. Since today's MCAD systems lack representation power for generic i-features and the capability to detect critical feature changes during the model development, they cannot guarantee even this basic precondition. This observation leads to the second research question that we posit in the introduction: which verification checks can one introduce in existing MCAD systems to ensure that the MCAD features in the final model satisfy the rationale for which they

were introduced?

Practical problems in using commercial MCAD systems

The modeling process in a feature-based MCAD system is based on the sequential application of *MCAD-feature commands*. The objective is to introduce the chosen MCAD feature into the model. First, the user selects the type of command that provides the required shape alteration; next, the user interacts with the system interface to carry out the different steps of the modeling command. The executed commands are stored in the feature history list and they can be edited (at any time and in any order) to modify the related modeling parameters.

MCAD-feature commands can be logically subdivided into local and global commands. Local commands (like rounds, chamfers, draft angles) are used to detail the local shape by altering single elements like edges and faces. Global commands (like cutouts, extrusions, holes, slots, etc.) are used to model the global shape of the component. Global MCAD-feature commands are usually applied by selecting a plane, drawing a profile on the plane and finally by defining the extrusion constraints. Global commands are the result of a sophisticated implementation of parametrized sweep and Boolean operations already existing in MCAD system before the feature-based approach. Today these have been merged into a sort of macro operation (Appendix A).

This approach is rather intuitive and allows to speed-up the modeling process. However, from the point of view of capturing, representing and maintaining the design intent, it has several drawbacks. First, MCAD-feature commands are defined and implemented with a shape-oriented instead of a semantic-oriented approach. Second, it is impossible to detect and manage MCAD-feature interactions with the consequence that the final shape of the model may not support the expected functions even though the needed features are present.

We now discuss an example to illustrate the nature of such issues. The example uses a very common feature in mechanical components, the hole feature. Beside additional details, like the presence of chamfer, counterbore or countersink, when a designer reasons about a hole, she will usually consider the hole axis as the main characteristic of the hole, together with its diameter and depth. These parameters are directly related to the function the hole is expected to support. During the design process it is fairly common to first introduce the holes in the component and to leave for a second step their qualification like

the specific desired values of the diameters. We may then expect the axis to be the central element for the definition of the hole feature. Surprisingly, none of the widely used feature-based MCAD explicitly represents the axis of the hole feature. Furthermore, when the hole at stake is a through hole, the information about this characteristic (through hole) is stored but the information about the hole depth is not.

From the software perspective, the hole feature is seen as a cylindrical shape whose axis is meant to be seen only as an implicit characteristic derived from the properties of the cylindrical surface. From the functional perspective, it should be the opposite: the cylindrical shape should be created by relying on the axis' position. The lack of explicit representations of some hole parameters introduces several limitations among which: the impossibility of locating and orienting the hole by explicitly referring to the axis position and direction; the impossibility of considering holes with the diameter value set to zero; the impossibility of explicitly dimensioning the hole depth.

This shape-oriented definition of features has further consequences on the possibility to verify feature interactions. In their present representation, MCAD features are defined in terms of the geometrical parameters required to generate the i-feature shape, typically a profile possibly with related dimensions, and an extrusion direction and limit. No information to keep under control the preservation of the generated geometry can be stored in the system. In the hole feature example, to implement some control mechanism one would need to add additional information and constraints, e.g., on the geometric elements to be preserved, like permanence of the bottom part of a blind hole, on the geometric condition to be maintained, like absence of blockage of a through hole, amount of required surrounding material for structural holes or persistence of the perpendicularity between the hole axis and entrance face for fastening holes.

Reclassifying features by their rationale in MCAD

Let us consider the hole feature presented earlier. There can be several reasons to introduce such a feature, e.g. to allow fastening, to introduce storage capacity, to reduce the component's weight. All these hole features have different rationales and we can isolate important differences by looking at their characterization. When a hole is introduced for fastening, say via screw and nut, the feature must be a through hole, there must be some open space around the openings and a perpendicularity condition among the hole axis and the entrance and exit surfaces (in contact with the strew head and the

nut) must be guaranteed. When the hole is introduced to ensure storage capacity, the feature must be a blind hole, there must be some open space around the opening and the volume must satisfy a given range. When the hole is introduced to reduce weight, the volume of the hole is maximized while some minimum depth of material must be ensured in every orthogonal direction of the hole internal surface. Even though the user is modeling all these features as holes, the constraints required by these different features implicitly carry information about the rationale for the hole feature introduction.
As an example, let us distinguish three types of hole i-features, that we isolate on the basis of their rationale (more precisely, their functionality):

- a joining hole (J-hole) is a hole whose rationale is that its realization allows fastening (e.g., via screws and nuts);
- a path hole (P-hole) is a hole whose rationale is that its realization allows some object to pass through that part of the product (e.g. a cable);
- a thinning hole (T-hole) is a hole whose rationale is that its realization reduces the weight of the product.

Clearly, these features are not disjoint: a J-hole is also a P-hole if the joining is done via screws and nuts as the screw has to pass through the hole. Moreover, a P-hole can be at the same time a T-hole. Second, to model a J-hole (via screws and nuts) or a P-hole one needs a relational property since the hole has to connect two distinct surfaces of the object hosting the hole. This property is not needed for the T-hole since it is irrelevant to this i-feature whether the hole is through or blind. Third, the rationales of these three types of hole constrain quite different parameters. In a J-hole the most relevant parameter is the hole axis as the alignment of the hole with the part to join is essential; in a P-hole the position of the axis might not be strictly constrained but the diameter is (more generally, the area of the section of the hole) as one needs to ensure that the material can pass through; in a T-hole the thickness of the material that bounds the hole and the information on the distribution of deformation forces are much more important than axis and diameter.
In order to capture the rationale behind these different types of holes and to support semantic reasoning about such entities, a MCAD system should be able to store and use the specific information that differentiate these types of i-features.
In present MCAD all these hole features are introduced via the same ‘hole feature’ command (or even via a generic linear ‘cutout feature’ command) and are associated with

the same set of information required to perform the geometric computation to insert the hole shape within the overall boundary representation of the model shape. No information dedicated to the rationale of the hole is stored, even information about relevant parameters, like the hole axis, is missing. Figure 1 shows part of the type of information stored in the MCAD model after a 'hole feature' command has been applied.

Hole feature	
plane	plane where the profile is located
profile	the circle defining the hole profile
extent type	type of the hole extension: blind, through next, through all, from-to faces
depth	hole depth. Undefined is the extent type is different from blind
side faces	the hole cylindrical face
bottom cap	the hole bottom. Undefined is the extent type is different from blind

Figure 1 – Portion of the information types stored in MCAD to represent a hole feature.

From an ontological point of view, the proper use of MCAD features requires a reclassification of these based on their semantics. This reclassification can be achieved by starting from the most general definition of i-feature, modeled in terms of primary and derived properties as seen for the J-hole, P-hole and T-hole cases, which allows to integrate the rationale of the i-feature within geometric modeling operations and property constraints.

Figure 2 shows a possible organization of the hole feature classes: each table represents an i-feature type and is defined by a name (grey cell), a set of primary properties (green cells), a modeling operator (orange cell), a set of derived properties (blue cell) and constraints (red cell). Subclasses inherit all the properties from the parent class and are specialized by means of additional properties.

The 'hole' feature table in Figure 2 defines the most general hole feature by means of three primary properties: axis, diameter and entrance face. The geometric operation required to model the shape of the feature is a linear sweep; the derived property, computed by the modeling operation, is the cylindrical surface; the most generic constraint a hole must satisfy is that the volume enclosed by the cylindrical surface will remain free; additional constraints may regard the range of the diameter value. The specification of the hole extension specializes the generic 'hole' into a 'blind hole' or a 'through hole', as represented by the linked tables in Figure 2. As for the 'blind hole' the

depth value is a primary property, in the ‘through hole’ the depth is a derived property that needs to be computed and that depends on the ‘extension’ property (through next, through all, etc.). The ‘fixing hole’ is a specialization of the ‘through hole’ with the additional constraint of the perpendicularity of the axis with respect to the entrance face.

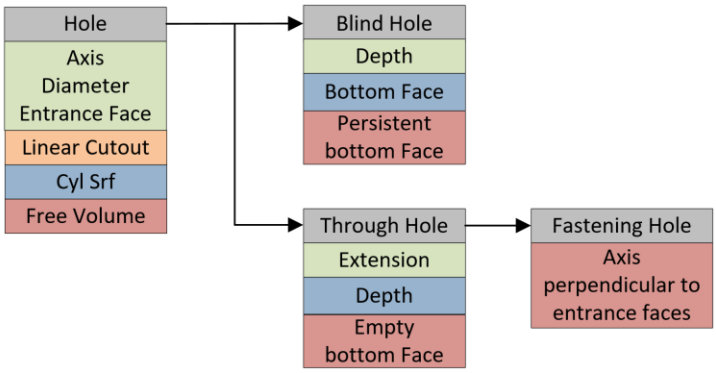

Figure 2 – Hole feature classification (the subclass relation goes from left to right).

The ontological definition of i-feature allows to identify the geometrical and topological entities that are relevant to the feature semantics; in order to capture the feature rationale, such geometrical and topological entities must satisfy specific constraints, i.e., a blind hole must have a persistent bottom face.

Some of the controls of the persistence of the required constraints can be easy to implement, like the geometrical computations on dimensional parameters of the feature (i.e. required functional volumes or perpendicularity condition); others may require more sophisticated approaches, mainly derived from feature recognition techniques, to check if the topological conditions (i.e. the presence/absence of topological entities) are preserved along the modeling process. Examples of such techniques date back to the late 1990s (Mandorli et al. 1997, Bidarra et al. 2000).

Revisiting MCAD features from the ontological viewpoint

The hole feature classification suggested in Figure 2 relies on the analysis of product modeling in the context of MCAD systems revolving around a core ontological question: ‘what is a hole?’. The aim of this analysis is to generate an ontology within the MCAD perspective and focused on the notion of hole. For our purposes, the ontology will be expressed in the OWL language (Hitzler et al. 2009), a standard language for ontology,

which we practically implement and visualize via the Protégé software[2], an open-source editor for ontology building and management.

An OWL ontology is a taxonomy of classes that represents the hierarchical organization of the types of entities relevant to the domain at stake and includes a collection of relationships of two kinds: object-property and data-property. Object properties are relationships that have as domain and range classes in the taxonomy (possibly the same) and are used to represent how elements of the classes are related to each other. For instance, 'being a component of' or 'being connected to' can be introduced as object-properties in the domain of assembly. Data-properties have as domain a class in the taxonomy and as range a datatype, that is, a predetermined set of entities (typical examples of datatypes are the integers and the alphanumeric strings).

In MCAD, the ontological question "what is a hole?" can be rephrased by: "what conditions should an MCAD object satisfy to be a hole?". As said, MCAD are essentially based on geometrical information, ontologically speaking in this setting a hole is a localised geometrical entity, namely, a cylinder with a given axis, diameter, depth, entrance face and empty interior. The latter is not a geometrical piece of information but is nonetheless managed by MCAD in general.

Due to the taxonomic structure of OWL ontologies, the ontology includes a class for each semantic feature that one can generate in MCAD, i.e., a class for holes (called Hole), a class for slots (Slot), etc. These classes form a hierarchy in the ontology under the high-level class MCAD_Entity. The ontology includes also a second set of subclasses under the high-level class MCAD_Component: each of these classes collects a type of element necessary for the existence of semantic features. Examples are axis, cross section and face. Thus, the MCAD semantic feature ontology has two main branches: MCAD_Entity and MCAD_Component (Figure 3a, left). In this paper we use only two classes of the MCAD_Entity hierarchy, namely Hole and Slot. For the MCAD_Component hierarchy, we consider six subclasses: Axis (necessary for holes), CrossSection (necessary for slots), CylindricalSurface (necessary for holes), Depth/Extension (necessary for both holes and slots), Face (necessary for both holes and slots), and SymmetryPlane (necessary for slots).

[2] https://protege.stanford.edu/

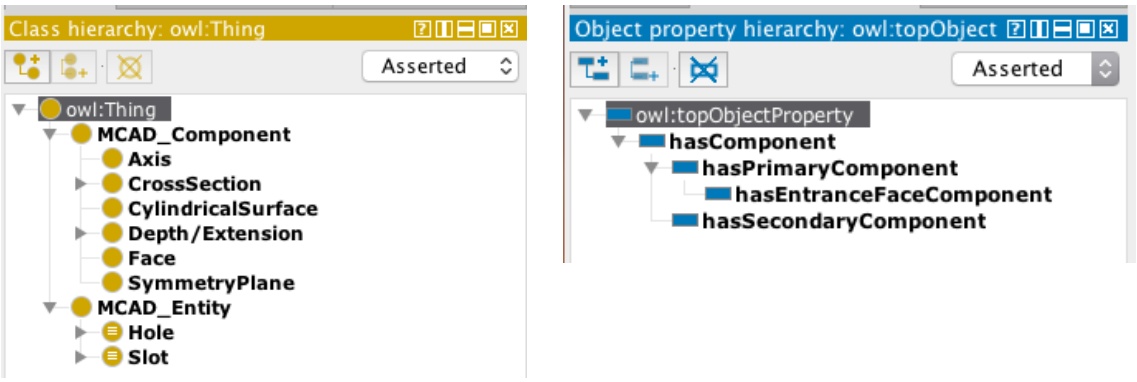

Figure 3a – The feature ontology organization: classes and object properties

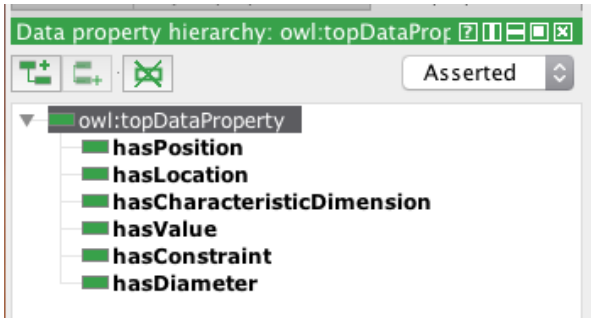

Figure 3b – The feature ontology: data properties.

Regarding the relationships that we use in the MCAD semantic feature ontology, the one called hasComponent holds between a MCAD_Entity and a MCAD_Component (Figure 3a, right). This relation has two subrelations: hasPrimaryComponent (further specialised in hasEntranceFaceComponent) and hasSecondaryComponent. The other relations are data properties as they connect a class of the ontology to a datatype (Figure 3b). We include in the discussion the following data properties: hasCharacteristicDimension, hasConstraint, hasDiameter, hasLocation, hasPosition, hasValue.

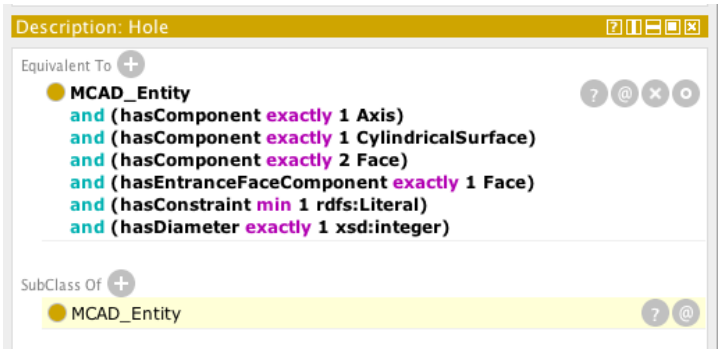

Figure 4a – The Hole class in the feature ontology.

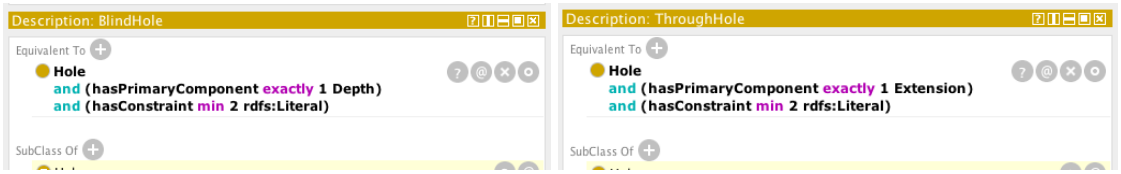

Figure 4b – The BlindHole and ThroughHole classes in the feature ontology.

A hole (Figure 4a) is defined as a MCAD_Entity with three characteristic relationships with MCAD_Component via the hasComponent relation: (1) it must be associated with one and only one axis, (2) it must be associated with one and only one cylindrical surface, (3) it must be associated with two and only two faces one of which, identified via the hasEntranceFaceComponent, is the entrance face. A hole must also be associated with a numerical value for the diameter, which is expressed via the hasDiameter data property. The Hole class is divided in two subclasses: BlindHole and ThroughHole (Figure 4b). Both of these classes inherit the characteristics of the parent class. Necessary conditions for a blind hole are the hole depth (a primary property) and the presence of a bottom face (a constraint). That is, a blind-hole is a hole such that (1) it must be associated with one and only one depth value, and (2) it must be associated with a bottom face.

When it comes to through holes, the characteristics are the extension (a primary property) and the presence of an empty bottom face (a constraint). That is, a through hole is a hole such that: (1) it is associated with one and only one extension value, and (2) the bottom face it is associated with is empty. The class ThroughHole is further specified into FixingHole, that is, through holes satisfying the further condition that their axis is perpendicular to the entrance face.

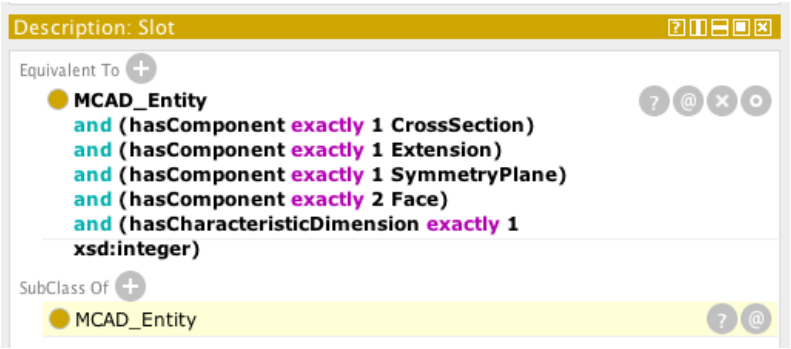

Figure 5 – The Slot class in the feature ontology.

Analogously to the Hole class, all MCAD semantic features are characterized by their relationships with MCAD_Component. For instance, the Slot subclass of the class

MCAD_Entity is defined as the class of MCAD entities that have these four characteristic properties: exactly one cross section, exactly one extension, exactly one symmetry plane, and exactly two guiding faces (Figure 5). Slots have also one characteristic dimension, which is a data property. The Slot class specifies into DoveTailSlot, T-ShapedSlot and U-ShapedSlot and the criterion for this distinction is the type of the cross section that each slot has as primary property (Figure 5). A graphical representation of the presented ontology is in Figure 6.

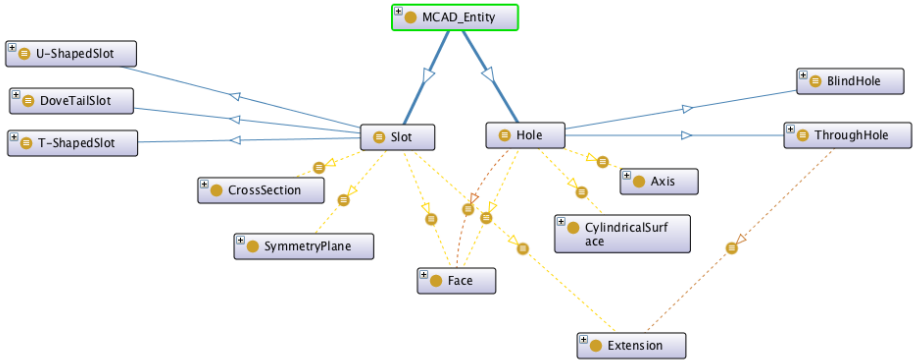

Figure 6 – Subsumption (solid arrows) and object properties (dotted arrows) of part of the feature ontology introduced in this paper.

Coverage and Limitations of the Approach

The ontology that we presented in the previous section shows how to identify core semantic elements in standard MCAD features, like holes and slots, via ontological analysis. The examples also show how this semantics can be encoded within the geometrical structure of existing MCAD. This work has two relevant consequences. It proves that a change of the geometrical and topological information organization suffices to embed truly semantic, and in particular functional, information into today's MCAD. It also shows that the introduction of verification checks across these geometrical and topological elements allows to verify that these MCAD features maintain the encoded functionality in the final MCAD model.

The most important contribution of our investigation is a methodology, exemplified with the hole and slot MCAD features, that allows:

(a) to identify the characteristic components of semantic features within the geometrical language of MCAD,

(b) to indicate which components are primary, i.e., ontologically capable to encode the associated functional feature,
(c) to build out of these features and components an ontological system that reliably organizes all this information, and
(d) to introduce verification mechanisms to check that a MCAD feature has the expected functionality in the model.

The semantic features that one can model in MCAD with our approach are the i-features relying on geometrical and topological information that can be extracted from data and parameters used by the MCAD commands, and whose characteristics can be expressed in terms of this information. Essentially, this tells us that the approach is limited to i-features that can be described or constrained in today's MCAD.
Interestingly, the approach seems applicable to features that, while geometry-dependent, are nonetheless out of today's MCAD modeling concerns, e.g., ergonomic features. In the case of ergonomic features, the initial step is to find the rules for the qualification of the product's ergonomic aspects. What is needed is a list of constraints on, e.g., the shape of a handle, the maximal/optimal surface contact between the parts, the distribution of weight and so on. These constraints are largely based on geometrical and topological information, but their validity depends on the sought ergonomic type of interaction. A deeper study of how to evaluate ergonomic features given the expected behavior of the parts is needed to verify the resulting ontology (Lewis and Narayan 1993).

Conclusions

In the early stage of CAD system development, improvement focused on geometrical algorithms and data structures. At that time, hardware performance was a bottleneck, for example for Boolean computation. The common intent of the subsequent developments in CAD, beside making the systems more efficient and stable from the computational point of view, was to improve the support of the decision-making activity. This was achieved by transferring part of the design process information from the user to the system via the use of features together with the possibility to manage geometrical parameters. Yet, due to the complexity of today's MCAD models, one may still prefer to re-create a model from scratch instead of attempting to adapting an existing one.
An ontological analysis of features that distinguishes their primary and secondary

components can provide the basis to develop appropriated knowledge-based functionalities for semantic design support. We showed one way to move forward relying on ontological analysis. What is needed is first a re-analysis and re-classification of features on the basis of their rationales, and then a re-organization of the different types of information related to the feature definitions. With this change it is possible to build general feature ontologies and to add semantic-based control mechanisms in the form of procedures that check the status of the feature primary properties and constraints during the model development.

Although, the approach does not give us a truly and comprehensive feature-based MCAD system, it makes possible to introduce and manage semantic, and in particular functional, information within MCAD features.

In the future, we plan to investigate further applications of our approach, and to include a cognitive analysis of the designer perspective in using geometric-based modeling tools. We believe that it is possible to use our feature ontology as the core of a modeling system where other aspects of the feature design rationale can be captured.

References

1. Bidarra, R., Bronsvoort, W.F., Semantic Feature Modeling, in Computer-Aided Design, V. 32, pp. 201–225, 2000.
2. Bodein, Y., B. Rose, and E. Caillaud. „Explicit Reference Modeling Methodology in Parametric CAD System.” *Computers in Industry* 65 (1):136–147, 2014. https://doi.org/10.1016/j.compind.2013.08.004
3. Camba, J. D., M. Contero, and P. Company. “Parametric CAD Modeling: An Analysis of Strategies for Design Reusability.” *Computer-Aided Design* 74: 18–31, 2016. https://doi.org/10.1016/j.cad.2016.01.003
4. Chandrasekaran, B. “Representing Function: Relating Functional Representation and Functional Modeling Research Streams.” *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 19 (2): 65–74, 2005.
5. Cheng, Zhengrong, and Yongsheng Ma. "A functional feature modeling method." Advanced Engineering Informatics 33: 1-15, 2017.
6. Garbacz, P., S. Borgo, M. Carrara, and P. E. Vermaas. “Two ontology-driven formalisations of functions and their comparison.” *Journal of Engineering Design* 22 (11-12): 733-764, 2011.
7. Hirtz, J., R. Stone, D. A. McAdams, S. Szykman, and K. Wood. “A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts.” *Research in Engineering Design* 13: 65–82, 2003.
8. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, Peter F. and Rudolph, S. OWL 2 web ontology language primer. W3C recommendation, 27 (1). p. 123, 2009.
9. Kitamura, Y., Koji, Y., and Mizoguchi, R. An ontological model of device function: industrial deployment and lessons learned. Applied Ontology 1, 3-4, 237–262, 2006.
10. Lewis, Winston G., and C. V. Narayan. "Design and sizing of ergonomic handles for hand tools." *Applied ergonomics* 24.5: 351-356, 1993.

11. Li, L., Zheng, Y., Yang, M., Leng, J., Cheng, Z., Xie, Y., Jiang, P., and Ma, Y. A survey of feature modeling methods: Historical evolution and new development. Robotics and Computer-Integrated Manufacturing 61 (2020), 101851.

12. Mandorli, F., Cugini, U., Otto, H.E., Kimura, F., Modeling with self Validation Features, in: Proceedings of Fourth ACM Symposium on Solid Modeling and Applications - Solid Modeling '97, pp. 88-96, Atlanta, USA, 1997.

13. Mandorli, F., and H. E. Otto. "Negative Knowledge and a Novel Approach to Support MCAD Education." *Computer-Aided Design and Applications* 10 (6): 1007-1020, 2013.

14. Mizoguchi, R., Y. Kitamura, and S. Borgo. "A unifying definition for artifact and biological functions." *Applied Ontology* 11(2): 129-154, 2016.

15. Otey, J., P. Company, M. Contero, and J. D. Camba. "Revisiting the design intent concept in the context of mechanical CAD education." *Computer-Aided Design and Applications* 15 (1): 47-60, 2018.

16. Otto, H.E., Mandorli, F., "A Framework to Support 3D Explicit Modeling Education and Practice." Computer-Aided Design and Applications, V.12, I.1, pp. 1-14, ISSN 1686-4360, 2015.

17. Pahl, G., Beitz, W., Feldhusen, J., Grote, K.H., Engineering Design: A systematic approach, Springer, 1996.

18. Sanfilippo, E., and S. Borgo. „What are features? An ontology-based review of the literature." *Computer-Aided Design* 80: 9-18, 2016.

19. Shah, J., and M. Mantyla. *Parametric and Feature Based CAD/CAM: Concepts, Techniques, and Applications.* New York: John Wiley & Sons, 1995.

Appendix A

In most of the commercially available feature-based MCAD, the feature commands can be classified as shown in Figure A1. The classification reflects the different types of modeling operations that are required to insert the feature into the model: first, features commands are subdivided into local modeling commands and global commands. Local commands introduce local modifications of the model shape, like rounding an edge (or a set of edges), and they do not involve volume computation; global commands are volume-based commands and they involve the use of Boolean operations. Among the global commands, one differentiates sweep based commands, that define the volume to be added or subtracted from the model shape by means of sweeping a profile along or around a curve, from offset based commands, that define the volume enclosed between two offset surfaces. The global commands can be used either to add or to subtract volume from the model shape; the next level in the classification differentiates between 'sum' operations (thicken and extrusions) and 'subtract' operations (shelling and cutout). The sweep-based operation can be further classified by means of the characteristics of the extrusion path: around an axis (revolution), along a curve (eventually a linear segment), through a set of sections (loft) or along a helix.

The classification is influenced by the technical evolution of the MCAD. ‘Engineering’ features, like hole, slot and rib, are implemented as specialized versions of generic commands, defined by means of specific profile shapes and extrusion direction. The classification makes evident that a loop feature is treated by the MCAD just like a generic cutout command with a predefined profile shape.

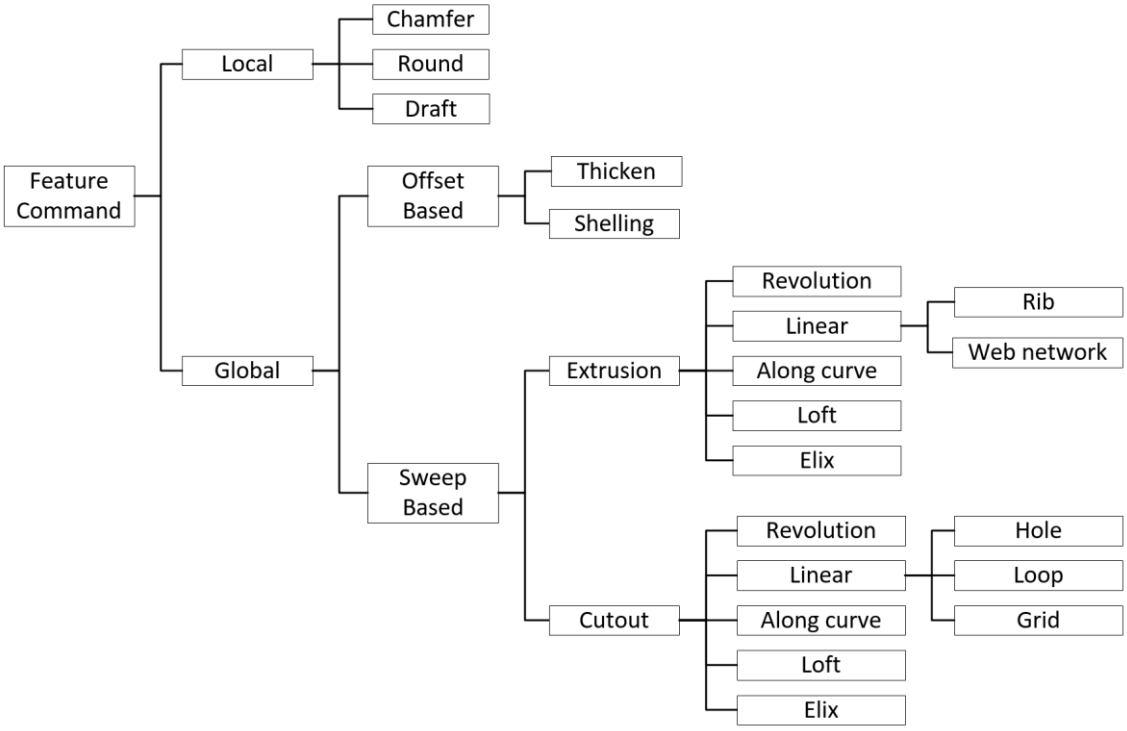

Figure A1. Classification of features in commercially available MCAD.