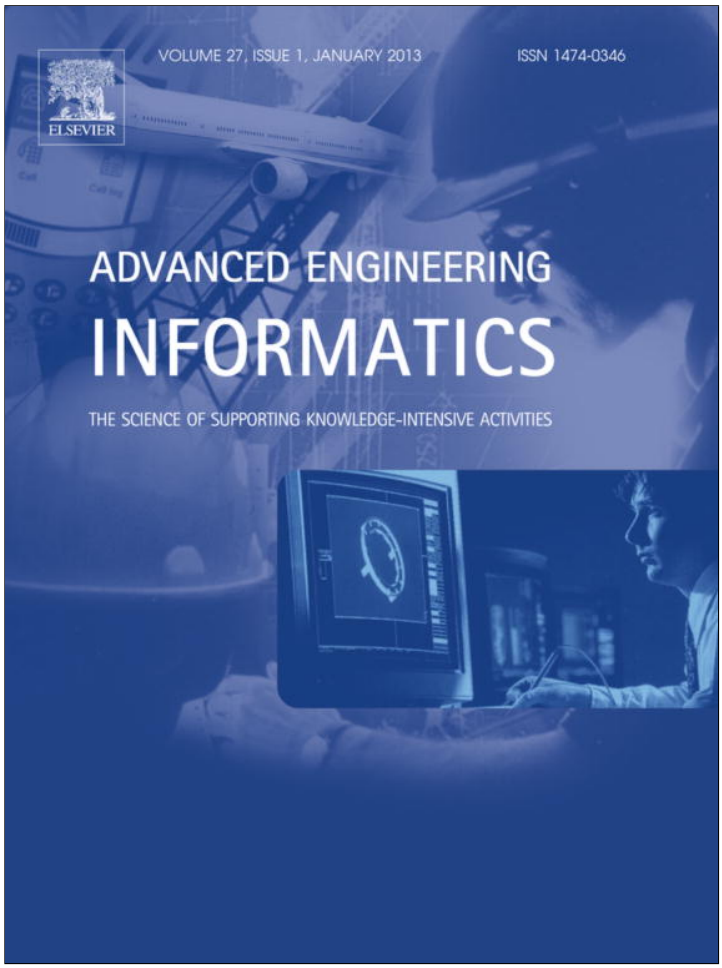


Advanced Engineering Informatics 27 (2013) 76–92

Contents lists available at SciVerse ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

Ontology-based feature mapping and verification between CAD systems

Sean Tessier, Yan Wang *

Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA 30332, United States

ARTICLE INFO

Article history:
Available online 21 December 2012

Keywords:
Computer-aided design
Feature
Interoperability
Ontology Web Language
Semantic Web Rule Language

ABSTRACT

Data interoperability between computer-aided design (CAD) systems remains a major obstacle in the information integration and exchange in a collaborative engineering environment. The use of CAD data exchange standards causes the loss of design intent such as construction history, features, parameters, and constraints, whereas existing research on feature-based data exchange only focuses on class-level feature definitions and does not support instance-level verification, which causes ambiguity in data exchange. In this paper, a hybrid ontology approach is proposed to allow for the full exchange of both feature definition semantics and geometric construction data. A shared base ontology is used to convey the most fundamental elements of CAD systems for geometry and topology, which is to both maximize flexibility and minimize information loss. A three-branch hybrid CAD feature model that includes feature operation information at the boundary representation level is constructed. Instance-level feature information in the form of the base ontology is then translated to local ontologies of individual CAD systems during the rule-based mapping and verification process. A combination of the Ontology Web Language (OWL) and Semantic Web Rule Language (SWRL) is used to represent feature classes and properties and automatically classify them by a reasoner in the target system, which requires no knowledge about the source system.

1. Introduction

One of the most common and costly problems impeding collaborative design environments is the difficulty involved in data exchange between heterogeneous engineering software systems. In practice, it is not uncommon to have to exchange computer-aided design (CAD) model data between different systems, as a company may use different software packages at different stages of the design process, or it may use a different one from their suppliers or business partners. Resources spent translating data between different CAD formats, reprocessing the data in different applications, redesigning due to information loss, or error checking and correction can become very costly. Neutral data exchange standards such as the Standard for the Exchange of Product Model Data (STEP) have been developed to facilitate CAD data exchange. Yet the standards have remained largely restricted to the representation of final geometry, causing the design intent portrayed through construction history, features, parameters, and constraints to be discarded in the exchange process. The inclusion of feature data in the STEP neutral format (ISO 10303) has been in discussion and under development in the past decade. During its implementation and industry practice, a neutral feature format to which all CAD systems must conform still has various practical issues. Some commercial translators, such as TTI's Accu-u-Trans [1] and TranscenData's Proficiency [2], offer feature-based conversion between CAD systems. They are often prohibitively expensive, and the numbers of supported systems are also limited. In such translators, problems often arise because of incongruous feature sets which require resolution from human user. A general, robust, and automated solution for feature-based exchange is still an active research area.

We differentiate two kinds of mapping processes in feature data exchange, each with their own advantages and disadvantages. One approach, shown in Fig. 1a, involves the *static mapping* of two class-level libraries to create a translator for one-to-one matches. This translator could then be used to directly translate an instance-level file from the source system to the target one. The other approach, shown in Fig. 1b, is a *dynamic mapping* process, where an instance-level file from the source system is compared to the class-level library of the target system, with an instance-level file being generated in the target system once the match is found.

The static mapping approach is currently used in industry practice. Based on the libraries of source and target systems, the analysis of feature similarity would only have to be done once. The feature classes then are stored as matching pairs, and the resultant translator could directly convert from one instance-level

* Corresponding author. Address: Woodruff School of Mechanical Engineering, Georgia Institute of Technology, 813 Ferst Drive NW, Atlanta, GA 30332-0405, United States. Tel.: +1 404 894 4714; fax: +1 404 894 9342.
E-mail address: yan.wang@me.gatech.edu (Y. Wang).

1474-0346/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.aei.2012.11.008

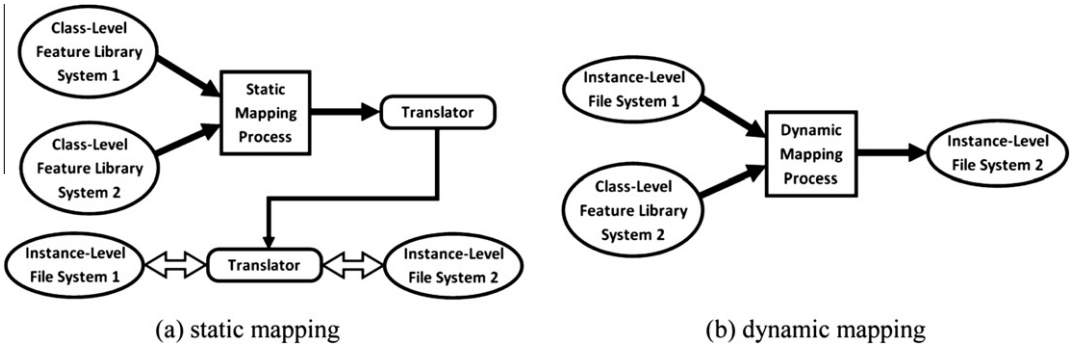


Fig. 1. Two approaches of CAD data translation.

representation into another. However, there are some obstacles with the static mapping for its general use. The first one is that it requires full access to both of the systems' class-level libraries and they must be kept up to date. This may be difficult when CAD vendors are not forthcoming about the proprietary information about their software. In addition, the one-to-one static mapping requires that one translator should be developed for each pair of CAD systems. Another major obstacle is the lack of a practical method to verify that each match between the source and target feature classes is valid. Unless there is a perfect one-to-one match, there is no guarantee that the use of the class pair with the highest similarity will result in a correct recreation of the feature. This is because the evaluated geometry of a feature is history-dependent. It relies on the prior geometry and the input parameters, both of which cannot be fully described in a general class-level sense. Therefore, the static mapping approach requires the two class-level libraries to be very similar to each other. Without verification tools, the only way to ensure a match would be human involvement, which offers few benefits over the ad hoc and case-by-case approaches.

Different from the static mapping, the dynamic mapping does not require the library of the source system in the process. This is helpful when new features or new CAD systems are introduced and no library is available. This is also useful to synchronize data in a client-server collaborative environment. When the client updates the parameters of a feature, the updated feature can be stored in an instance-level document and uploaded to the server. The dynamic mapping process then can locate the specific feature in the old instance-level document and update the parameters. More importantly, with the dynamic mapping, it is possible to implement a feature verification process after a target feature is created. When working with instance-level source and target files with geometry data simultaneously during the mapping, the recreated geometry after the translation can be checked and compared to the original one. Additionally, because the dynamic mapping only requires the library of the target system, it can be integrated into the data import process of any CAD system without knowledge of the source system library. It is helpful for individual companies to protect their intellectual property such as their unique features. Yet the dynamic mapping would be inherently less efficient than the static one, because similarity assessment has to be made every time a file is translated, which is the cost of the flexibility it offers.

In this paper, we propose an approach to support the dynamic mapping of features such that geometry can be verified automatically. This approach uses a new feature data structure that stores both the instance-level feature definition and boundary representation (B-Rep) data from the source system as the input. The information is converted to a neutral format based on a shared base CAD ontology that includes the fundamental and widely accepted concepts of geometry in the entire knowledge domain of CAD. The fundamental geometric information of features can then be automatically mapped to the features defined in the target systems according to their local ontologies and some necessary and sufficient conditions coded as rules. The output of the rule-based mapping is the feature data defined in the target systems. The proposed approach addresses the limitations of the purely semantics based mapping methods in other approaches for feature-based data exchange. We are able to store all the available feature data from the source system using a set of very basic ontology classes. Thus information loss would only occur when the target system cannot support specific feature data. The reproduced geometry can be verified.

The major contribution of the paper is a new and robust approach for CAD ontology generation by fully storing feature data including geometry such that verification can be performed in parallel during the feature matching process. It uses ontologies to characterize features via geometry-oriented operations that convert the semantics of feature definitions to specific shapes, which is different from other ontology-based feature mapping approaches that simply define feature ontologies as classes that require a certain set of attribute types and focus only on semantic level comparisons of domain specific feature class definitions. Those approaches would require all CAD feature data to fully conform to a prescribed standard, which is overly restrictive from the perspective of CAD vendors. In contrast, our approach enables feature information sharing only at the fundamental level of two-dimensional sketches and B-Rep data. By applying sets of rules that a particular target system uses, feature-level information is inferred locally from the fundamental elements of geometry and topology. The rules also provide additional inference and verification capability for matching local features that are uniquely defined beyond relying on semantic similarity of class definitions. Our approach requires a less restrictive standard data format without mandating a shared set of standardized features. It provides more flexibility that individual CAD systems would need, improves the robustness of feature information matching with the verification, and reduces the amount of human input that is required.

In the rest of the paper, the most relevant work in feature mapping for interoperability and their limitations are reviewed in Section 2. Section 3 discusses our approach with the ontology representation and rules defined to model features. Section 4 describes the implementation and demonstration of our approach. Section 5 will conclude with a summary and discussions of future work.

2. Background

2.1. Relevant work of feature mapping between CAD systems

Historically, feature mapping has been researched to translate from one domain to another [3,4], particularly from design to manufacturing and process planning [5,6]. As feature-based modeling

become dominant, feature mapping has been applied to convert from one CAD system to another. The core question is the lossless feature information exchange between systems via certain media or forms other than their native formats. Editable Representation (E-Rep) [7,8] developed by Hoffmann et al. was an early attempt at exchange of construction information. It specified models as a sequence of feature insertion, modification, and deletion processes. Han and colleagues [9–11] captured the construction history by a journal file created by CAD systems. This journal file contains a record of the commands utilized by the user, which is then converted into a non-STEP neutral macro format. More recently, the neutral macro format was updated to support geometry-based data to avoid the problem of part reference based on creation order [12], although the issue of persistent naming [13–17] still exists. Rappoport et al. [18–20] represented features using a B-Rep structure. A concept of feature rewrite was used to compute the changes in geometry before and after a feature operation so that the history of construction can be exchanged. The research focused on the retention of geometric information and has been implemented in the commercial translator package Proficiency. Li et al. [21] established a real-time collaborative design environment based on the use of neutral modeling commands. Application programming interfaces (APIs) of the source and target systems were used to exchange construction information across networks in real-time through the neutral commands. The above approaches achieve feature interoperability by recording the construction history in the source system and using it as the instruction guide for the target system, instead of exchanging actual model data.

A different approach is to model and exchange the actual model data. The Project ENGEN (Enabling Next GENeration design) [22] involved the extension of the STEP standard to more than the pure geometric information. The focus of project ENGEN was the exchange of geometric constraints. The exchange of two-dimensional (2D) data containing constraint information was demonstrated. Some research has also been done to translate feature data including constraints across heterogeneous systems using files with the extensible markup language (XML) format. The import and export are accomplished by APIs of CAD systems. Examples are the work from He et al. [23,24], Zhang and Luo, [25] and Sun et al. [26]. In these efforts, feature mapping was done manually.

Research has also been sought to add constraint data to the most recent STEP standard to convey design intent. Kim et al. [27] proposed to enhance the STEP standard with exchangeable construction history and shapes with parameterization and constraints. They noted a common problem that most researchers have experienced when attempting to create exchange programs using CAD systems' APIs is that the APIs of commercial CAD software are not primarily intended as an interface for model exchange. The future research of an ontological approach for the semantic mapping of modeling elements between CAD systems was emphasized, such as the early work of Patil et al. [28] that captured feature semantics with ontology. Ontology is a formal representation of a set of concepts, their properties, and the relationships between those concepts within a given domain. Such common language is essentially a neutral format to enable translation between domains, which reduces the number of translators and helps resolve semantic differences. Some tools that have also been created to support inferencing in ontologies are highly useful when working with heterogeneous data sets.

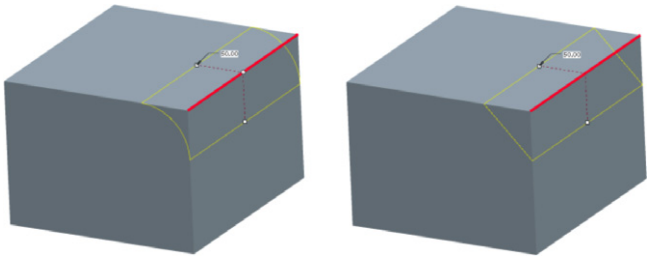
The use of ontologies has become increasingly favored in approaches for exchange of the complete semantics of feature data. Seo et al. [11] exchanged the history of feature construction with a macro-parametric approach through an ontology using the F-Logic format. Wang and Nnaji [29] demonstrated that the semantics of features in different CAD systems can be captured through the ontological approach and inferencing tools can be used in automatic reasoning and mapping. Dartigues et al. [30] showed that data exchange between a CAD ontology and a computer aided process planning ontology through the use of a common domain ontology can be achieved, even though the CAD ontology only contained geometric data instead of the complete construction history and parameters. Jayaram et al. [31,32] illustrated the ontology-based interoperability between product design and assembly simulation domains, where cross-domain translation can be done through a bridge ontology only if one-to-one matches of semantics are identified. In our previous work [33,34], the feature mapping was based on semantics of feature definitions and also relied much on CAD system APIs.

2.2. Importance of comparing both feature semantics and geometry in mapping

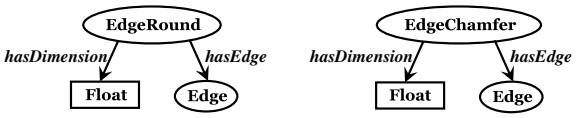
The feature mapping approach proposed in this paper also takes advantage of ontological tools. However, it differs from the above ontology based approaches by examining the individual instance-level features from both viewpoints of feature semantics and the instantiated geometry. Our approach focuses more on describing features as interactions between their basic parameters and the evaluated geometry, instead of simply defining features with some presumed standard feature definitions. This approach contains a generalization of feature representation developed for the automated feature mapping [33,34]. Here, we employ rule-based reasoning techniques similar to the ones proposed by Henderson et al. [35,36] for feature recognition. Instead of applying between CAD and other application domains, the use of inference rules here is to find instances of individual features in different CAD systems from the information of both feature semantics and geometry. Each feature instance is recognized in comparison with the local ontology in the target system's feature library, before any automated feature mapping occurs. This is an important distinction from other feature mapping approaches, which rely solely upon semantics of feature definitions. Our approach more accurately identify a feature instance as an object of a particular feature class even when there are semantic differences in feature definitions. This is a critical characteristic because comparing feature definitions alone does not always give one-to-one matches between heterogeneous systems.

Other approaches that use purely semantics mapping just compare attribute labels and graph structures of feature definitions. When a match is found, all instances are directly translated according to the target definition. When a match is not found, it would attempt to determine the best match by running similarity calculations. The semantic similarity is calculated at the class level between source and target features. What these approaches are lacking is the fundamental shape information that a feature represents at the instance level, which can cause ambiguity. Two simple examples are used here to illustrate the semantic ambiguity issue. In Fig. 2, an edge fillet from one system is mapped to another. Suppose we had no prior knowledge that the round feature in the target system generates the same geometry as the fillet feature in the source system. The semantic approach would have to resort to attribute type comparison. In the target system, a round and chamfer feature have the same set of attribute types, as shown in Fig. 2b. Both features take a float value to describe the dimension, and a single edge as a reference. If purely based on the feature definition, the mapping process would be unable to determine whether round or chamfer the fillet should be mapped to, and would have to rely on the human user, who ultimately has to compare the actual geometry of some feature instances and make the decision.

Another example that could be problematic involves two types of round features with slightly different options within the feature definition. Suppose the source system has a 'blend intersection'



(a) Round feature vs. Chamfer feature



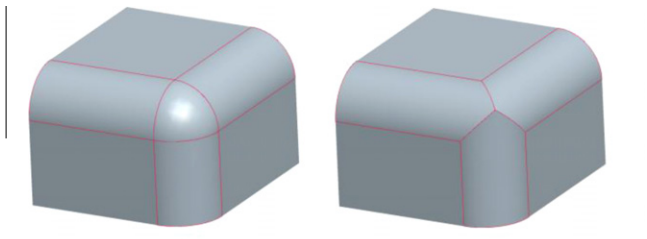
(b) The similar semantics of feature definitions

Fig. 2. Ambiguity of feature semantics between round and chamfer.

attribute stored as a Boolean value whereas the target system has no such attribute. As shown in Fig. 3, the geometries at the edge intersection area are different, with and without this single option. The mapping process could simply provide the match between the two round features instead of providing a better solution (e.g. a combination of features to capture the blend intersection information). The system cannot verify if the translated feature recreates the original geometry. Again, the user would have to check instance-level geometry manually. In both examples, the mapping would require a great deal of user input and negate the benefits of using ontologies.

To overcome this dilemma, it is necessary to examine the ambiguity problem from more than just a semantic viewpoint. It is essential to build the connection between the semantic input and the expected geometric output, in order to reduce the human user involvement and better automate the mapping process. By adding B-Rep data into the ontology-based feature representation, we can give the computer a basic means to verify geometry and enable feature mapping with particular instances. However this is still not enough to provide an efficient solution because it would require a large number of tests and feature evaluations to eliminate wrong matches and recreate a specific geometry. The ideal approach would emulate a human translator who conceptually understands features in the target system and predicts the geometric output without actually having to create it completely. The person might not know exactly how all attributes should be mapped in the target system, but he/she could still reduce the search to only those features that can generate similar geometry.

In general, data interoperability problems are due to *structural* and *semantic* heterogeneity. Structural heterogeneity is the incompatibility because of the different data structures being used,



(a) with blend intersection (b) without blend intersection

Fig. 3. Ambiguity in feature semantics between two round features with different intersection options.

whereas semantic heterogeneity is the incompatibility that arises from naming or terminology differences. Most interoperability approaches mainly focus on the semantic heterogeneity because of its prevalence. It is important to note that structural heterogeneity is also a major problem and must be addressed. It is fairly easy to automatically map feature data from one system to another when equivalent features are structurally identical with some semantic differences through the use of ontologies, such as when one-to-one matches are identified. The real challenge is determining how to automatically map features that are equivalent but are defined with different data structures. This task is usually delegated to a human user to provide solutions. The existing ontology-based approaches in Section 2.1 focused more on the semantic heterogeneity when one-to-one matches are available. In contrast, our approach also addresses the process of identifying equivalent features automatically in cases when no such straightforward matches are available.

3. The proposed ontology-based feature mapping and verification

When ontologies are applied to integrate data and specify semantics, three types of approaches can be taken [37]. They are single-ontology, multiple-ontology, and hybrid-ontology approaches. The single-ontology approach specifies a single global ontology for the entire knowledge domain. The multiple-ontology approach specifies a separate local ontology for each source of data. The single global ontology approach is too rigid since it requires that all sources of data have the exactly same vocabulary, whereas the multiple local ontology approach is too unstructured since the lack of a shared vocabulary among the ontologies means that mapping data between different systems is difficult and inefficient. The hybrid-ontology approach resolves the issues of both single and multiple ontology approaches by combining them. In this approach, a globally shared base vocabulary contains the basic concepts of the knowledge domain, which is complemented by local domain ontologies for more complex concepts with respect to individual sources of data. The use of a shared base ontology allows information to be exchanged in a vocabulary that is domain independent, from which each local ontology can be interpreted.

Here, we take the hybrid-ontology approach to enable interoperability of CAD feature data. As shown in Fig. 4, the proposed process of data exchange would work as a dynamic mapping process. First the data for a specific model instance from the source system are extracted from the system format and parsed into the shared global base ontology format. The data contain the construction sequences of features in the form of a proposed three-branch hybrid CAD feature model, which stores all relevant data with both feature semantics and geometry. Once the data have been exported, they can be used by any CAD system that is able to import data from the shared base ontology format. During the import process, the reasoner of the target system examines each feature being imported from the shared base ontology file, and runs a series of tests. For each imported feature, these tests check which feature subclass in the target feature library contains rules that have not been violated, and therefore possibly recreating the original geometry specified by the target features. Once these tests have eliminated all feature subclasses which cannot reproduce the source geometry, then more traditional similarity mapping can be applied to resolve the remaining semantic problem if there is any. In cases of structural heterogeneity, additional rules are needed to check if some types of feature data are superfluous, if additional information is required and what form it must take, or if a feature has no equivalent and cannot be reproduced. With the B-Rep information stored in the neutral file of shared base ontology, the generated

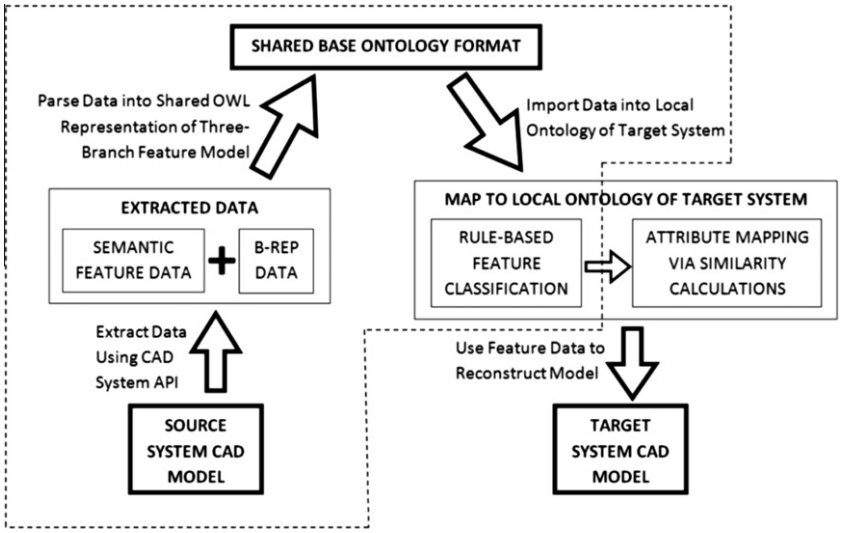


Fig. 4. The proposed general mapping and verification approach for feature interoperability.

target geometry could be used to verify if the match is correct. Once a match has been found, the mapping can be stored, and the translation could be performed without the testing from then on.

A shared global base ontology for the feature-based CAD domain is necessary for efficient dynamic mapping and data exchange between systems. However, it should be noted that the global base ontology only creates a set of base concepts and properties which all features in any system can *use* in their definitions. It should not specify how features should be defined. In the CAD domain, the base concepts and properties that all systems can understand turn out to be geometry and topology related. Therefore, the shared base ontology contains these common terminologies. In parallel, we also need the local ontologies that simply store combinations of those base concepts to define the features for each CAD system. Creating a single global ontology that goes beyond the base for all CAD systems is practically not possible, as it would require each system to agree on a standard shared set of features and a universal way to define them. It is very difficult to make decisions such as how the global ontology should build feature class hierarchies, what the supported minimum set of features must be, and how detailed each feature definition must be, because there is no "right" way to define a feature and it is a matter of individuals' and companies' opinions. History has shown that establishing a universally accepted industry standard for one neutral format is an intricate task.

Each CAD system will have its own ontology that can work together with the shared base ontology. Local ontologies have the advantage of allowing each system to store data as they see fit, and are not restricted to a limit number of feature types. In our approach, feature data are parsed directly into the shared base ontology, and then mapped into the specific local ontology by rule-based inferencing. The shared global base ontology acts as the instance-level neutral format. The class-level library of features is stored in the local ontology of the target system, where the system-specific features can be represented as combinations of concepts from the shared global base ontology. By expressing feature data through this ontology-based representation, feature semantics can be stored and processed using various software tools.

Instead of the purely semantic mapping approach, our new approach uses a set of rules stored within the local ontology to derive which features are capable of reproducing the source feature geometry before any semantic mapping similarities are made. These rules represent the necessary and sufficient conditions that must be true for a given feature type to have a valid representation. They uniquely identify features by the output types that must be generated given particular sets of input, which is how features are distinguished at a conceptual level by human users. These rules are similar to a rule based approach to feature recognition. Because these rules are written in a class-level representation, they should closely match the internal code and algorithms that the CAD system uses to determine what shape the feature generates, given a particular set of input. By using rules to define geometric validity, features from other systems can be identified as matches even if there are significant semantic differences. In order to support this new approach, we need to develop a feature model that incorporates B-Rep data, as described in Section 3.1.

3.1. Three-branch hybrid feature model

The three-branch hybrid feature model proposed here is to model a CAD feature in terms of the individual settings and parameter values selected by the user during feature definition. As shown in the general diagram in Fig. 5, there are three major branches of data in defining each feature. They are *reference attributes*, *parameter attributes*, and *B-Rep operations*. Reference and parameter attributes are explicitly distinguished to better classify their data types in the definition. The reference attributes pertain to information that is necessary to the definition, but is defined externally outside the feature. This can be a pointer to an existing reference datum, face, edge, vertex, or sketch data for sketch-based features. Conversely, the parameter attributes refer to information that belongs exclusively to the feature, such as an option or numerical value that the user specifies, which is independent of other features. This distinction is important because the parameter attributes can be compared and converted directly by using the established

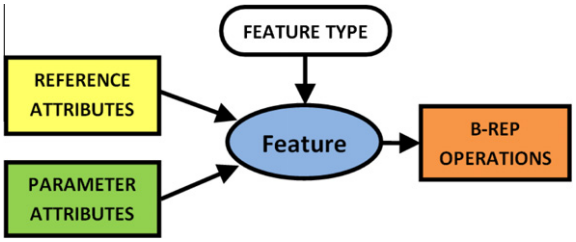


Fig. 5. General structure of the three-branch hybrid feature model.

datatypes such as floats and integers, whereas the reference attributes will likely have different identifiers in different systems. For example, consider a feature that rounds an edge. The parameter attributes identify the type of rounding and the value of radius. The reference attribute is the specific edge identifier, which would differ between CAD systems without a persistent naming convention. These two sets of attributes make it possible to describe a class-level representation of the feature, as the types and numbers of reference attributes should be independent of the specific values entered at the instance-level.

The third branch of the feature model represents the B-Rep operations, which stores the changes the feature makes to the geometry at the instance-level representation. This cannot be described in a class-level representation because the changes to the B-Rep can only be determined when the reference and parameter attributes have specific values. These instance-level data are also used as a verification measure. Some feature definition rules can be used to verify the number and type of certain B-Rep entities. The extra *feature type* attribute in the hybrid feature model may be used to store the possible names of the feature in some systems as aliases (e.g. extrusion and protrusion) so that a match is recorded for ease of use in the future. The feature type attribute is unnecessary for feature classification. But it allows the target system to know what type of feature it has been mapped to so that future instances of the same feature type can be recognized without going through the rules again.

The three-branch hybrid feature model serves as a template to describe a feature using the shared base ontology language. The goal of the hybrid feature model is to strike a balance between prescribing a neutral format through which all features can be described and maintaining the expressiveness and individuality of each CAD system. Base classes for three branches are defined and shared globally. The unique local ontologies of individual CAD systems specify the types and quantities of class instances described by each feature. Descriptions of the classes from the shared base ontology will be provided in Section 3.2. This template is necessary to reduce the amount of semantic and structural heterogeneity as much as possible without constraining the expressiveness of the feature definitions from the CAD systems. This has two major advantages over other ontology-based feature models which concentrate only on feature semantics. First, the added expressiveness allows the comparison of more factors than the names and datatypes of parameter and reference attributes, because the base ontology can contain more geometry-oriented concepts for various common parameter classes, such as radius and length, which will ensure that semantic ambiguity is avoided as much as possible. Second, the definitions of reference attributes and B-Rep operations can be potentially standardized because they are only based on geometric concepts that have been well established and are fairly universal amongst different CAD systems. Specifying a standard structure and nomenclature for these reference attributes and B-Rep operations does not restrict the expressiveness of features and in most cases would be a direct process. In other words, our approach does not mandate that standard features follow shared definitions. Rather, the existing feature definitions are parsed into a shared format with a basic geometric-level vocabulary. Although the persistent naming will still be a problem with reference parameters and B-Rep operations, this issue is likely to be resolved by comparing the geometry itself to determine the equivalency [16,38]. But it is out of this paper's scope.

Fig. 6 shows an example of the hybrid feature model for an extrude feature. The reference attributes are the reference plane used to define the orientation and the sketch. Parameter attributes determine whether it creates a solid or surface model, the option to set the depth type, the depth value (if blind or symmetric is chosen), whether to flip the direction of the sketch normal, and whether the extrude feature is being used to add or remove material (i.e. Boolean union or difference).

Similarly, Fig. 7 shows an example of a revolve feature, where the sketch is rotated about an axis to create geometry. As such, it has the same reference attributes, with the addition of an axis of rotation to determine the line about which the sketch is rotated. It has fewer parameter attributes, because revolves are specified using the angle of rotation rather than depth.

The two examples shown here only have the basic classes of B-Rep operations at the class level. More detailed operations specifically on geometric and topological elements depend on the instance level information. Nevertheless, with this generic structure of what information is included for each feature, all features can be defined in a similar vein.

More rigorously, the syntax model of the three-branch hybrid feature is defined as a directed labeled graph $\Sigma = \langle \mathcal{N}, \mathcal{E}, l \rangle$, where $\mathcal{N} = \{\mathcal{N}_f, \mathcal{N}_r, \mathcal{N}_p, \mathcal{N}_b\}$ is a set of feature nodes $\mathcal{N}_f$, reference attribute nodes $\mathcal{N}_r$, parameter attribute nodes $\mathcal{N}_p$, and B-Rep operation nodes $\mathcal{N}_b$. $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{T} \times \mathcal{N}$ is a set of edges connecting nodes with the corresponding edge types $\mathcal{T} = \{T_s, T_a, T_g\}$ where T_s = '*specialization*', T_a = '*aggregation*', and T_g = '*general association*'. $l : \mathcal{N} \mapsto \mathcal{L}$ is a label function that defines the label of each node where $\mathcal{L} = \{\mathcal{L}_C, \mathcal{L}_R, \mathcal{L}_V, \mathcal{L}_E\}$ is a set of labels for the nodes with four types, including textual characters $\mathcal{L}_C$, identifiers or references $\mathcal{L}_R$, real,

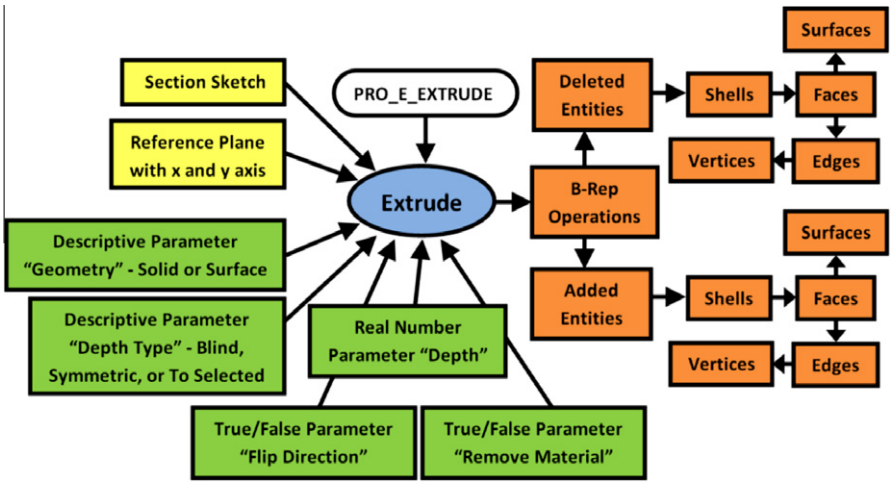


Fig. 6. Three-branch hybrid feature model of extrude.

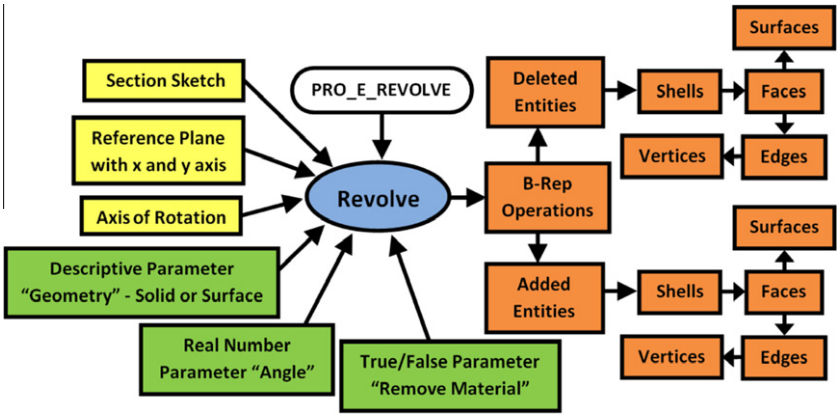


Fig. 7. Three-branch hybrid feature model of revolve.

integer, or Boolean values $\mathcal{L}_V$, and geometric and topological entities and operations $\mathcal{L}_E$. Particularly, for node n with its label $L = l(n)$, $L \in \mathcal{L}_C$ if $n \in \mathcal{N}_f$, $L \in \mathcal{L}_R$ if $n \in \mathcal{N}_r$, $L \in \mathcal{L}_V$ if $n \in \mathcal{N}_p$, and $L \in \mathcal{L}_E$ if $n \in \mathcal{N}_b$.

With the necessary information elements, the next step is to generate the shared base ontology language through which these three-branch CAD feature models can be described, as discussed in the next section.

3.2. Shared base CAD ontology

The shared base CAD ontology should carefully define classes that represent the very basic concepts and properties in CAD. To create the shared base ontology, we made use of the Ontology Web Language (OWL) [39], because it adds more vocabulary to describe properties and classes than the Resource Description Framework (RDF) [40] used in our previous research. It also has a variety of reasoning tools readily available. The formal specifications of the OWL language were based on the $\mathcal{SH}$ family of description logic, particularly $\mathcal{SHIQ}$ and $\mathcal{SHOQ}(\mathbf{D})$, which include the Attributive Concept Language with Complements ($\mathcal{ALC}$) by intersection, union, and complement operations, universal and existential quantifiers, and transitive property; the class and property hierarchy support ($\mathcal{H}$); inverse function ($\mathcal{I}$); enumerated instances ($\mathcal{O}$); generalized cardinality restrictions ($\mathcal{Q}$); and the use of datatype properties ($\mathbf{D}$).

The semantic model of the feature ontology is defined as $\Theta = \langle \mathcal{K}, \mathcal{P}, \mathcal{H}, Q, c, r, i \rangle$, where $\mathcal{K} = \{\mathcal{C}, \mathcal{D}\}$ is a set of feature concepts, with $\mathcal{C}$ as the subset of classes and $\mathcal{D}$ as the subset of datatypes. $\mathcal{P} = \{\mathcal{P}_C, \mathcal{P}_D\}$ is a set of properties, with $\mathcal{P}_C \subseteq \mathcal{C} \times \mathcal{C}$ as the subset of object properties and $\mathcal{P}_D \subseteq \mathcal{C} \times \mathcal{D}$ as the subset of datatype properties. $\mathcal{H} = \{\mathcal{H}_C, \mathcal{H}_P\}$ is the set of hierarchical relations, including class hierarchies $\mathcal{H}_C \subseteq \mathcal{C} \times \mathcal{C}$ and property hierarchies $\mathcal{H}_P \subseteq \mathcal{P} \times \mathcal{P}$. $Q = \{\wedge, \vee, \neg, \rightarrow, \forall, \exists\}$ is a set of logic operations and quantifiers that form set-based logic statements, including conjunction $\wedge$, disjunction $\vee$, negation $\neg$, implication $\rightarrow$, universal quantifier $\forall$, and existential quantifier $\exists$. $c : \mathcal{P} \mapsto \mathbb{Z}$ is a cardinality function that maps a property to one or many integer values. $r : c(\cdot) \times Q \times \mathcal{P} \times \mathcal{K} \mapsto \mathcal{K}$ is a restriction function that narrows down the class domain. $i : \mathcal{K} \mapsto I$ is an interpretation function that maps the semantic model to a $\mathcal{SHOIQ}(\mathbf{D})$ interpretation I.

With the notation of description logic [41], $i(\mathcal{C}) = \Delta^I$ and $i(\mathcal{D}) = \Delta_D^I$ where Δ and Δ_D are the interpretation domains of object and datatype respectively, and ^{I} is the $\mathcal{SHOIQ}(\mathbf{D})$ interpretation function. $i(A) = A^I$, $i(A \cup B) = A^I \vee B^I$, $i(A \cap B) = A^I \wedge B^I$, and $i(\neg A) = \Delta^I \setminus A^I$, where $A, B \in \mathcal{C}$. With individual instances $a \in A$ and $b \in B$ where $A \in \mathcal{C}$ and $B \in \mathcal{K}$, as well as an object property $(a, b) \in R \subseteq \mathcal{P}$, $r(\exists, R, B) = \{a \in A | \exists i(b), i((a, b)) \in i(R) \wedge i(b) \in i(B)\}$, $r(\forall, R, B) = \{a \in A | \forall i(b), i((a, b)) \in i(R) \rightarrow i(b) \in i(B)\}$, and $r(c(\cdot), R, B) = \{a \in A | c(b), i((a, b)) \in i(R) \wedge i(b) \in i(B)\}$. It is obvious that $r(\exists, R, B) \subseteq A$, $r(\forall, R, B) \subseteq A$, and $r(c(\cdot), R, B) \subseteq A$, where $\subseteq$ denotes the class hierarchy.

3.2.1. Class hierarchy

The shared CAD global ontology is created such that the classes represent the fundamental geometry-oriented concepts that should be universal amongst different CAD systems. The major class hierarchy is shown in Fig. 8, where the arrows illustrate 'is-a' relationships and are used to show subclasses as specialized subsets of the main classes. Sibling classes are disjoint with one another and mutually exclusive.

The shared base CAD ontology classes in Fig. 8 are described as follows. The *PartFile* class stores the fully exported part from a CAD system, and is composed of a set of reference attributes, such as the coordinate system and three orthogonal reference planes, and references to a series of features. An instance of this class contains object properties that point to specific reference attributes and feature class instances. The *Feature* class acts as a general place

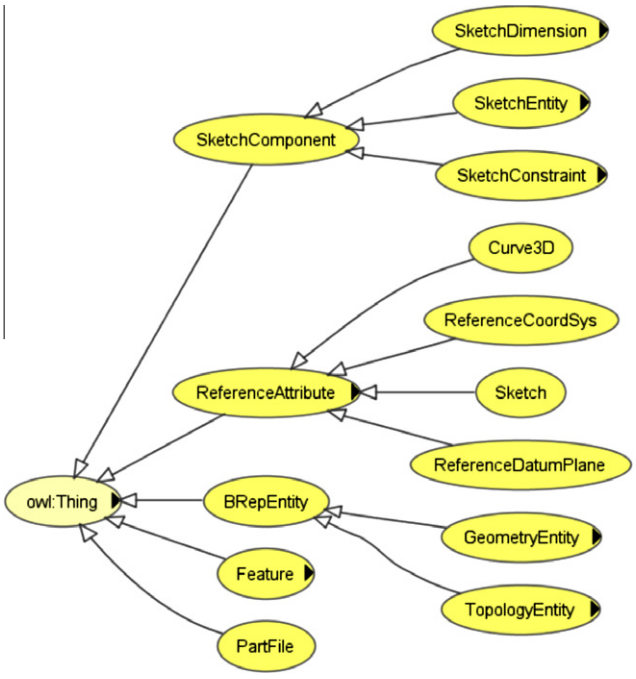


Fig. 8. Major classes of the shared base CAD ontology.

holder, with only three basic rules indicating that it must have at least one reference attribute, parameter attribute, and B-Rep operation. Further subclasses of feature are not defined in the shared base CAD ontology. Rather, it is the local ontologies for individual CAD systems that will define them to correspond to their particular feature sets. The *ReferenceAttribute* class describes the various types of references, such as sketches, reference planes, coordinate systems, and existing part reference types, used in a feature definition. An object of the *Sketch* class is composed of parts of the *SketchComponent* class, which includes fairly universal concepts of entities, dimensions, and constrains. These *SketchComponents* are items that are used to define a sketch and thus related to a *Sketch* object via object properties. The parameter attributes are not stored as classes because they are composed of datatypes already supported by OWL, such as integers, floats, Booleans, and strings. They are handled through datatype properties. Finally, the *BRepEntity* class is composed of the well established concepts of geometry and topology entities as *TopologyEntity* and *GeometryEntity*.

There are more subclasses that are not shown in Fig. 8. The example subclasses of SketchConstraint are SamePointConstraint2D, ColinearConstraint2D, PerpendicularEntitiesConstraint2D, HorizontalEntityConstraint2D, VerticalEntityConstraint2D, PointOnEntityConstraint2D, EqualRadiiConstraint2D, etc. The example subclasses of SketchDimension are ArcToArcHorizontalTangentDim2D, ArcToArcVerticalTangentDim2D, LineLengthDim2D, LineToPointDim2D, RadiusDim2D, DiameterDim2D, etc. The subclasses of TopologyEntity include Shell, Face, Loop, Edge, Coedge, and Vertex. The subclasses of GeometryEntity include Surface, Curve, and Point. Furthermore, the subclasses of Surface may include PlanarSurface, SphericalSurface, ConicalSurface, ToroidalSurface, and SplineSurface. The subclasses of Curve may include LinearCurve, EllipticalCurve, HelicalCurve, and InterpolatedCurve (e.g. B-SplineCurve, ParameterSpaceCurve, SurfaceIntersectionCurve) to represent the specific types of each.

Table 1
SketchEntity subclasses and their properties.

Domain	Property	Range	Cardinality
PointEntity2D	*hasXCoord*	[float]	=1
	hasYCoord	[float]	=1
CoordSys2D	hasPoint	PointEntity2D	=1
CoordAxisEntity2D	hasStartPoint	PointEntity2D	=1
	hasEndPoint	PointEntity2D	=1
ConstructionLineEntity2D	hasStartPoint	PointEntity2D	=1
	hasEndPoint	PointEntity2D	=1
LineEntity2D	hasStartPoint	PointEntity2D	=1
	hasEndPoint	PointEntity2D	=1
ArcEntity2D	hasCenterPoint	PointEntity2D	=1
	hasStartAngle	[float]	=1
	hasStartPoint	PointEntity2D	=1
	hasEndAngle	[float]	=1
	hasEndPoint	PointEntity2D	=1
	hasRadius	[float]	=1
CircleEntity2D	hasCenterPoint	PointEntity2D	=1
	hasRadius	[float]	=1
EllipseEntity2D	hasCenterPoint	PointEntity2D	=1
	hasXradius	[float]	=1
	hasYradius	[float]	=1
ConicEntity2D	hasStartPoint	PointEntity2D	=1
	hasEndPoint	PointEntity2D	=1
	hasShoulderPoint	PointEntity2D	=1
	hasConicParameter	[float]	=1
PointArray2D	*hasN_Points*	[integer]	=1
	hasPoint	PointEntity2D	=N
PolylineEntity2D	hasPointArray	PointArray2D	=1
SplineEntity2D	hasPointArray	PointArray2D	=1
	hasStartAngle	[float]	=1
	hasEndAngle	[float]	=1

The *SketchComponent* class has three main subclasses, grouped as *SketchEntity*, *SketchConstraint*, and *SketchDimension*. The subclasses of *SketchEntity* are listed in Table 1. The first column of the table lists the names or domains of the subclasses. The second column lists the object properties and datatype properties of the subclasses. Object properties link an individual to another, whereas datatype properties link an individual to a specific type of value. The datatype properties are italicized and their ranges of value domain in brackets. Note that these properties must apply to every instance of the entity. The cardinality specifies the number of instances that are subject to the property. Therefore, each class is connected to the other classes through various properties with 'has-a' relationships. The way in which features use properties in their definitions is a major source of heterogeneity. Yet the standard set of basic properties defined in the proposed base ontology is generic enough and acceptable for all CAD systems, because they are no more than the fundamental elements of geometry, topology, sketch dimensions and constraints which formed the foundation of modern CAD systems.

3.2.2. Property hierarchy

The property hierarchy also needs to be established. Here, the properties related to B-Rep operations are used to illustrate the hierarchical structure. Every B-Rep property is described as a subproperty of the *hasBRepOperation* parent property, which has the *BRepEntity* class as its range. This parent property splits into the *hasTopologyOperation* and *hasGeometryOperation* subproperties, with ranges of the *TopologyEntity* and *GeomertyEntity* classes respectively. These subproperties are then divided into creation and deletion subproperties, which are then further specialized to refer to the various subclasses of topology and geometry entities. For example, denoted as $createsPlanarSurface \subseteq createsSurface \subseteq createsGeometry$ where $createsPlanarSurface, createsSurface, createsGeometry \in \mathcal{P}$, *createsPlanarSurface* is a subproperty of *createsSurface*, which itself is a subproperty of *createsGeometry*. The ranges also follow this hierarchy, as the range of *createsPlanarSurface* is the *PlanarSurface* class, which is a subclass of *Surface*, which in turn is a subclass of the *GeometryEntity* class. B-Rep operations were divided into creation and deletion operations. Because without a persistent naming convention, B-Rep entities can only be uniquely identified by their specific definition. Tracking changes is much simpler if the modification is always by a deletion followed by a creation.

Notice that parameter attributes are slightly more problematic than the reference attributes. Parameter attributes are defined using datatype properties. Instead of linking the feature to an instance of a particular class by object properties, datatype properties are linking to a specific value of integer, float, string, or Boolean value. Similar to the reference attributes, a major problem of semantic mapping arises when more than one property is using data from the same range. This problem is compounded in parameter attributes because features often have multiple parameters that use the same datatype and a datatype contains very little conceptual information. Fortunately, datatype properties can also have subproperties, so long as the subproperty has the same datatype as its range. Therefore, multiple subproperties can be used to distinguish between different parameter properties that use the same datatype. The problem then becomes a matter of creating a parameter attribute subproperty hierarchy that defines the common types of parameters. It is important to note, however, that even establishing a set of common parameters may not prevent semantic heterogeneity. For example, one program may describe the distance of an extrude feature using a float parameter called 'depth' whereas another program uses 'length', 'D1', or any other names. In this case, setting a common vocabulary may become difficult. This problem has to be resolved through collaboration to

determine a specific set of concepts, and through the clever use of subproperties. For example, the *hasDimensionAttribute* property which stores float values could include subclasses such as *hasLengthTypeDimension* and *hasAngleTypeDimension*, which are used to distinguish values expressed in units of length from those expressed in units of degrees or radians. The *hasLengthTypeDimension* property could have further subproperties such as *hasDepth*, *hasRadius*, *hasDiameter*, *hasThickness*, *hasLength* and other common ways in which lengths can be recognized. The benefit of this approach is that more information is available to work with than simply the datatype during semantic similarity mapping. When attempting to compare a feature that uses *hasLength* to one that uses *hasDepth*, we could move up one level and find that both are instances of a property which measures length. That is, a higher similarity can be found other than simply checking two float values. Other relationships could also be included such as simple conversions. Common terms could be related through conversion rules. For example, diameter equals two times radius, or arc length equals radius times angle (in radians). In summary, the main idea here is to use property hierarchies to provide more information about the stored data.

Other general principles of practices include using Boolean values when there is a choice between two options such as a check box, and using string values when there are multiple options such as a dropdown menu. Integers should only be used when dealing with options that require integers, such as when features that copy parts or create patterns. Using integers to store options types (e.g. in an enumerated list) or Boolean operation should be avoided, because it makes the information ambiguous, and is counterproductive for data interoperability. The goal is to present data in a way that can be easily understood both by a human and ontology reasoner with as little knowledge of the source system as possible.

The last property to note is the *feature type* attribute in the three-branch hybrid feature model assigned to allow for faster mapping once a valid match has been already established. This *feature type* attribute is stored as a simple string datatype property. Once a feature match has been established via the dynamic mapping process and verified as correct, a new rule could be created to automatically map all features with the same value for the *feature type* property directly to the matching target feature, forgoing the computations used in dynamic mapping.

With the shared base ontology established, the framework for exporting CAD data from any system into a neutral format is complete. The task of the export process only requires the data from a given CAD system be parsed into instances of the shared base ontology classes. To complete the data exchange process at the target system side, the feature data must be mapped from the fundamental elements of geometry and topology with the shared base ontology format into the feature classes defined by the local ontology of the target CAD system. This classification process is described in the following section. Nevertheless, it should be acknowledged that the shared base ontology described in this paper is not meant to be definitive or all-encompassing, but simply to illustrate how a shared base ontology language could be constructed for the CAD domain. It would be more appropriate for a definitive CAD base ontology to be established by an organization such as the National Institute of Standards and Technology (NIST) with input from the various CAD vendors.

3.3. Rule-based feature classification in local ontologies

The shared base CAD ontology does not specify any details about how individual features are defined, except for the basic information of 2D sketch and B-Rep geometry. To allow for automated mapping of any feature instance defined according to the shared base ontology, the local ontology of the target system should contain a feature hierarchy with the *Feature* class from the shared base ontology as the top-most base class. From there, the local ontology should define subclasses corresponding to different families of features, in an organization that is the most logical for the target system, effectively creating a comprehensive feature set.

There is no correct or single way to define features. Each CAD system can have its own internal class hierarchy. For instance, Fig. 9 shows one definition of feature class hierarchy (similar to

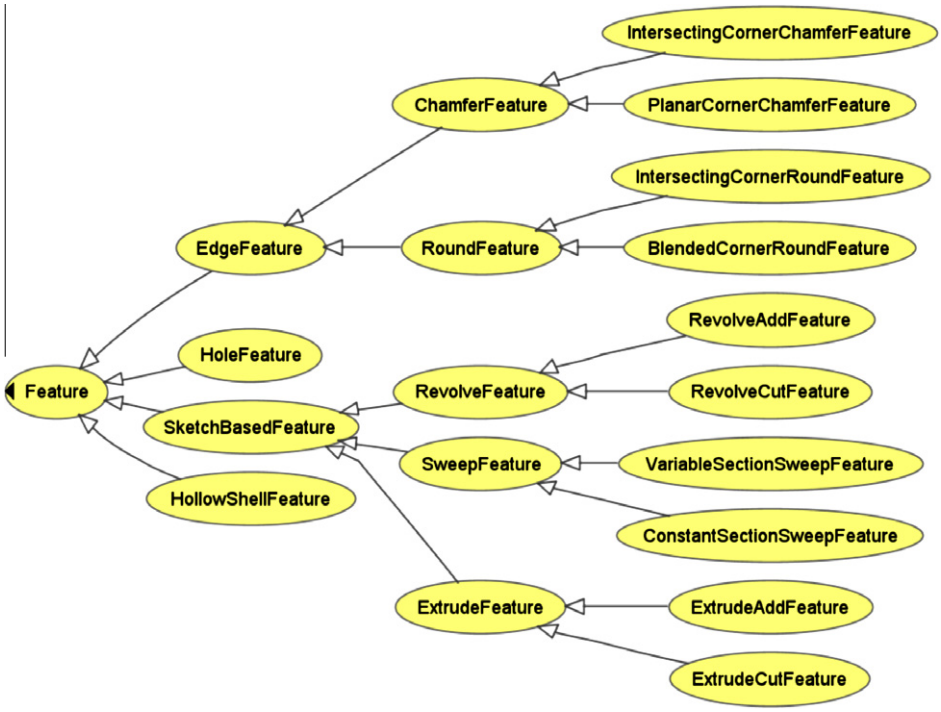


Fig. 9. Example A of feature hierarchy.

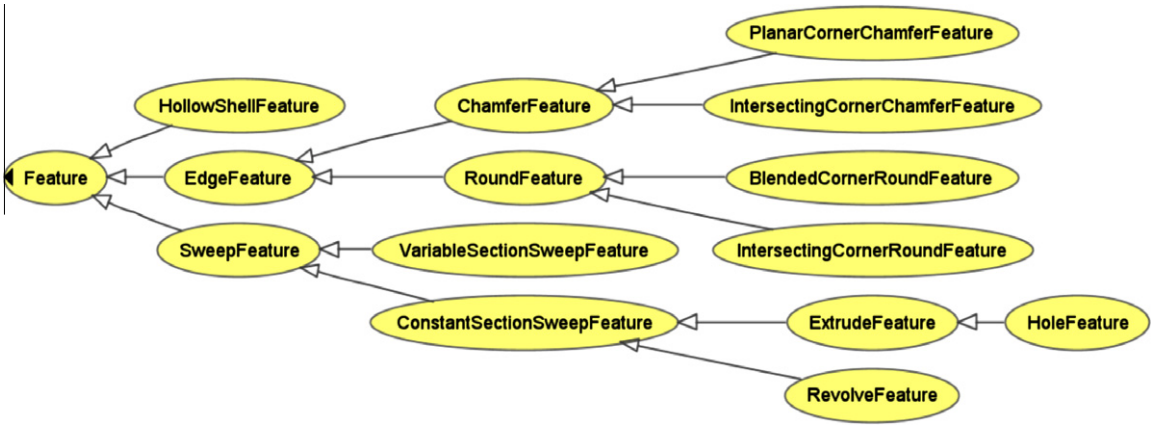


Fig. 10. Example B of feature hierarchy with extrude and revolve as subclasses of sweep.

the one in Pro/Engineer), where all features that require a sketch are grouped as *SketchBasedFeatures*, and the features that effect edge geometry are grouped as *EdgeFeatures*. Additionally, revolve, extrude, and sweep are treated as separate features. This feature hierarchy may not be suitable for other systems. For example, consider that a constant section sweep takes a 2D sketch and extends it along a designated path. One can consider the extrude and revolve feature as simply special cases of the sweep feature. An extrude can be considered a sweep along a linear path normal to the sketch, while a revolve is a sweep along a circular path. In that case, it is conceivable that a CAD system may want to treat the extrude and revolve features as subclasses of the constant section sweep. Similarly, very simple hole features can be considered a special case of an extrude feature in that the sketch is replaced with a circle of given radius. If a CAD system were set up this way, it would be better to use a local ontology with a feature hierarchy displayed in Fig. 10. The groupings in this example do not work with Pro/Engineer, since all sweep features require a path in Pro/Engineer and an extrude could not be a subclass without that information. Yet in a third system, it may not deem necessary to code extrude and revolve features separately if the sweep included easy options to generate linear and circular paths, such as the one proposed by Dartigues et al. [30] and shown in Fig. 11. Features are separated into the ones that affect volume directly and those that deal with face transitions. This type of hierarchy would also work efficiently with systems that were built based on the constructive solid geometry. The additive features would be used to represent features that use the Boolean union, whereas the subtractive features use Boolean difference and intersection.

The goal of local ontologies is to conceptualize features in a way that an ontology reasoner can understand. A set of necessary and sufficient rules can be defined such that they must always hold true for a specific feature. For example, consider a simple solid extrude feature created by projecting a 2D sketch linearly in a direction normal to the sketch plane. Every entity in the sketch is going to create a surface and there will be two planar surfaces on each end. Every line entity in the sketch is going to create a planar surface, each circle or arc will create a cylindrical surface, and each

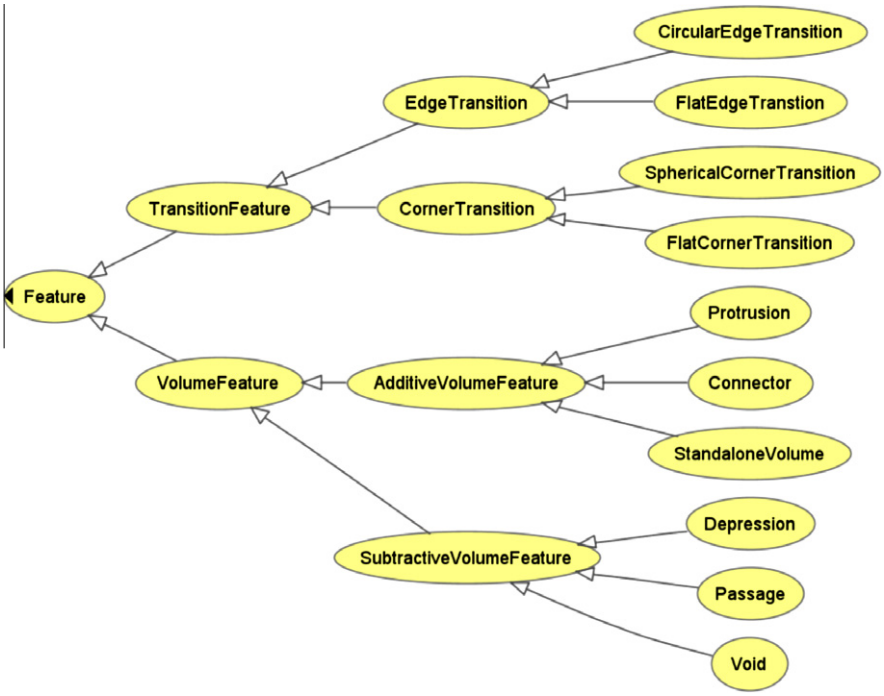


Fig. 11. Example C of feature hierarchy (proposed by Dartigues et al.).

spline in the sketch will create a spline surface. Therefore a generalized rule for the extrude feature including the case when the solid extrude intersects another object is that *for N sketch entities, extrude will create at most N + 2 surfaces*. By correlating specific sketch inputs to specific geometry output values, rules of creating all feature types allow a reasoner to automatically determine which classes the feature being translated belong to, without relying on exactly matching semantic data. All features have a set of rules that define them conceptually and they must obey, so establishing these rules as necessary and sufficient conditions allows semantically unmatched instances of a feature to be automatically classified in the target system, if these conditions are met. In essence, the comprehensive set of rules that define a feature in its local ontology are the same rules that are being implemented internally within the specific CAD system to automatically convert the user input to the geometry that is displayed.

The use of feature hierarchies helps facilitate feature mapping and classification based on the local ontologies tailored to individual CAD systems. To classify a feature that is imported into the target system, a series of rules are used to progress down the class hierarchy in the local ontology. Use the hierarchy in Fig. 9 as an example. The feature from the source system is imported as a general and unclassified instance of the top level *Feature* class. From there, the feature is tested with certain rules to determine which of the next level of subclasses it can belong to. For instance, the test to see if a feature is a *SketchBasedFeature* would check the number of *hasSketch* properties. If $Cardinality(hasSketch) \geqslant 1$, it is assigned to the subclass, and no longer tested for the other branches.

Rules are specified such that statements of features can be derived. Generally speaking, given feature class A, feature concept B, and property R, if a restriction function $r(\exists, R, B) = A_1$ or $r(c(\cdot), R, B) = A_1$ can be established, then $A_1 \subseteq A$. That is, the feature class A can be further classified as A_1. For instance, in the above sketch feature example, $a \in Feature$, $b \in SketchEntity$, and $(a, b) \in hasSketch$. If a restriction $r_1(c_1(\cdot), hasSketch, SketchEntity)$ where $c_1(hasSketch) = \{1, 2, \ldots\}$ exists, then a *Feature* is also a *SketchBasedFeature*.

To further classify the *SketchBasedFeature*, more tests are run on every instance of that class. The test for a basic *ExtrudeFeature* checks every instances of *SketchBasedFeature* class to see if 'the number of surfaces is equal to the number of sketch entities plus two'; 'the number of planar surfaces equal to the number of line entities plus two'; 'the number of conical surfaces equal the total number of circle and arc entities'; and finally that 'the total number of spline surfaces equals the total number of 2D polyline, spline, ellipse, and conic entities'. If all of the above is true, then the *SketchBasedFeature* instance becomes a member of the *ExtrudeFeature* class. The respective cardinality restrictions are $r_2(c_2(\cdot), hasSurface, Surface)$ where $c_2(hasSurface) = c_1(hasSketch) + 2$; $r_3(c_3(\cdot), hasLineEntity, LineEntity2D)$ where $c_3(hasLineEntity) = \{0, 1, \ldots\}$ and $r_4(c_4(\cdot), hasPlanarSurface, PlanarSurface)$ where $c_4(hasPlanarSurface) = c_3(hasLineEntity) + 2$; $r_5(c_5(\cdot), hasCircle, CircleEntity2D)$ where $c_5(hasCircle) = \{0, 1, \ldots\}$, $r_6(c_6(\cdot), hasArc, ArcEntity2D)$ where $c_6(hasArc) = \{0, 1, \ldots\}$, $r_7(c_7(\cdot), hasConicalSurface, ConicalSurface)$ where $c_7(hasConicalSurface) = c_5(hasCircle) + c_6(hasArc)$. The corresponding existential restrictions are 'sketch-based feature has surfaces', 'sketch-based feature has sketches', 'sketch has line entities', 'sketch-based feature has planar surfaces', 'sketch has circle entities', 'sketch has arcs', and 'sketch-based feature has conical surfaces'. Notice that restriction functions return classes. Therefore, the conjunction of multiple rules for one classification step is by recursively assigning restrictions so that the resulted subset of sketch-based feature is a more specialized extrude feature. Similarly, the rules to verify the instance of the *RevolveFeature* class are those to see if specific sketch entities are correlated to specific surface types. They include 'line entities that are perpendicular to the axis of rotation would create planar surfaces and all other lines would create conical surfaces'; 'circular arcs centered on the axis of rotation would create spherical surfaces and those that are off-center would create toroidal surfaces'; and 'all other 2D curve entities should result in spline surfaces and all points not on the axis of rotation should create circular curves'. Members of the *SweepFeature* class would have to be composed of surfaces that are correlated to the swept path, with planar surfaces on both ends and a curve duplicating the path for every 2D point entity in the sketch.

These rules could still be far from being necessary and sufficient conditions for classifying a feature. They are a simplification of the more comprehensive process needed for feature recognition in the setting of commercial software. For example, conditional statements with more if-then branches would be needed when the feature is interacting with existing geometry, where union, intersection, or difference is applied. If additional detailed information is needed, new rules can be defined. For instance, a reasoner could not only check if the right types of surfaces are created, but also confirm that the surface is defined with the right parameters. However, such rules would require very detailed knowledge of how the feature is locally defined. In summary, the feature recognition rules would have to emulate the internal feature creation and validation rules as used in the target system. We argue that the rule-based tests can be done with any feature, as features are always defined in a CAD system by the predictable way such that user input is transformed into geometry based on some internal rule schemes. Those rules already exist and are used in individual CAD systems.

Let us use the *EdgeFeature* subclass from Fig. 9 as the second example of the rule-based classification. The test to determine if a feature is a member of this subclass, rules would determine if all the instances of reference attributes of the feature are members of only the *ReferenceEdge* class. It would also check if 'every *ReferenceEdge* selected is accompanied by at least one surface creation and the selected edge is deleted from the B-Rep by the feature operation'. To distinguish between *RoundFeatures* and *ChamferFeatures*, the types of surfaces would again be tested. To be considered an instance of the *RoundFeature* class, there must be 'a conical surface creation for every linear curve selected', 'a toroidal surface creation for every circular curve selected', and 'a spline surface created for every elliptical and interpolated curve selected'. Similarly, to be a member of the *ChamferFeature* class, there must be 'a planar surface creation for every linear curve', 'a conical surface for every circular curve selected', and 'ribbon like spline surfaces for all other curve types'. To test for the different corner blending options for both of these features, the rules would simply check for additional surfaces. For instance, 'if more planar surfaces are created than linear curves were selected, then the *ChamferFeature* will most likely have corner planes', and 'if there are spherical or additional spline surfaces in a *RoundFeature*, then there is likely some corner blending'. Once the list of possible feature matches have been narrowed down in this way, the mapping process becomes much easier. More rules and rigorous tests could be performed to ensure that the curvatures of the resulted surfaces match those of the edges, and there could be conditional rules when the geometry of the part causes exceptions.

The third example to illustrate is a *HoleFeature*. The best test to run on any hole feature is to check if 'all conical surfaces created share the same central axis, and all planar surfaces are centered on and normal to that axis', regardless of whether the hole has a counter bore, countersink, or tapered end. If this rule is satisfied, the number and types of surfaces can be tested to determine what kind of hole options were used. These rules are available because holes are easily distinguished in commercial automatic feature recognition software. Again, it is important to stress that the rules represent the knowledge to create these features in each of the target

CAD systems and have been encoded into the software programs. Therefore they are ready to be applied if this approach of ontological mapping and verification would be employed.

The major advantage of our approach is that if the classification rules properly reflect the necessary and sufficient conditions for the target feature, it should always work provided the target system has a feature that can replicate the geometry of the source feature. The simple tests described above require very little calculation but are capable of partially determining which feature subclass the imported feature is. In an ideal case, the feature rules need to be as rigorous and complete as those used to define and generate geometry inside the target CAD system. Highly details of how the CAD system operates must be known. Defining the complete set of rules could be a costly and laborious task for those who do not have the direct access to this knowledge base. It makes reverse engineering by a third party very difficult. Such an ideal case is likely to be only implementable by the CAD vendor itself. As a result, this approach of ontological mapping from the shared base to local ones preserve the uniqueness of each CAD program. There is no need for a CAD vendor to share proprietary information with others, so long as the CAD companies agree on the shared base ontology format and are willing to develop a hierarchy of features with a series of tests to evaluate and check for each type of feature class.

There are potential cases when our approach will fail to find the match in the local system. It will fail when there is no feature in the target system that adequately resembles the shape concept conveyed by the original feature, but such an instance would cause problems for any semantic based approach as well. In spite of this, if the local ontology uses a branching hierarchy as in the examples above, this approach still has a benefit over other semantic mapping processes, because it classifies the source feature as the lowest feature subclass that the rules proved was valid, which has narrowed down the number of choices for manual mapping. Another case when this approach would fail is when one system uses a compound feature, which would have to be replicated by more than one feature in another system. For example, Solidworks allows the option to include a draft angle in their extrude feature definition which tapers all sides in by the given angle. Pro/Engineer has no such option in their extrude feature. To replicate the design intent, the users of Pro/Engineer have to first extrude the shape, then use a separate draft feature. Here, the rule-based approach would fail because no version of the Pro/Engineer extrude would be able to create B-Rep in which surfaces are not perpendicular to the sketch plane. In this case, the classification rules stop after listing the feature as an instance of the *SketchBasedFeature* class and would be unable to proceed. A purely semantic based approach may be able to classify it as an *ExtrudeFeature*, but it too would be unable to reconcile the difference between the shape types. Nevertheless, with the full B-Rep information included in the exchange file in our feature model, the geometry of any feature that is incapable of being mapped could still be recreated by inserting "dummy" surface that are not defined parametrically, which is also an approach taken by some commercial translators.

Ontology of source and target features may be changed or updated as time evolves. If the change is initiated by the source system, the effort of ontology update is negligible because the shared base ontology is B-Rep geometry focused and most likely it remains unchanged. The source system only needs to ensure that it can export B-Rep geometry of new features in order to make the share base ontology complete. For the target system, classification rules may need some update to better match the newly introduced source features. Otherwise, "dummy" surface as B-Rep geometry is still a viable alternative. If the change is initiated by the target system, the local classification rules in the reasoner need to be updated so that the mapping from generic features to specific local ones is valid. The change however does not affect the source system.

4. Implementation and demonstration

To demonstrate that the proposed approach is viable, we implemented the core part of the approach as outlined by the dotted line in Fig. 4, where feature information required for the three-branch hybrid model is extracted from the source system, data are parsed into the shared base ontology format, and then individual features are recognized and classified using local ontological rules defined in the target system. In our implementation, the Semantic Web Rule Language (SWRL) rules [42] are used to implement the classification rules in combination with the OWL representation. The additional step of attribute mapping via similarity calculation module is a pure semantic mapping that was done in our previous work [33,34]. In this work, a shared base CAD ontology was created using Protégé-OWL [43]. PTC's Pro/Engineer CAD software was used to demonstrate export of features into the shared base CAD ontology format. A sample set of feature classes were created to demonstrate a local ontology feature hierarchy. The SWRL rules were implemented in Protégé-OWL's SWRLTab and run using the Jess rule engine [44].

OWL describes classes with at least one necessary and sufficient condition as a defined class, while those without are described as primitive classes. Therefore, if one can represent a feature in a local CAD ontology as an OWL class, and properly define the set of necessary and sufficient conditions based on restrictions of global ontology properties, then any feature that has the set of properties satisfying those conditions could automatically be inferred as a member of that class, regardless of the source system. This implementation serves only as a proof of concept, as the export process only supports single feature models, and the rule-based classification is severely limited by the lack of non-monotonic reasoning in OWL and SWRL. Despite these limitations, the classification of extrude features from Pro/Engineer to a sample local ontology of another system demonstrates that this approach is viable given a robust rule-based reasoning language.

4.1. Shared base ontology

Classes and properties in the OWL format are fairly easy to create using the Protégé-OWL interface. As stated in Section 3.2.1, the five categories of classes and subclasses were created. They are the *Partfile* and *Feature* classes used to store the exported part and the top of the feature tree, the *ReferenceAttribute* class used to define the various types of reference attributes, the *SketchComponent* class, where various sketch entities, constraints, and dimensions are defined, and finally a *BRepEntity* class, where the different types of B-Rep concepts used by the ACIS format are defined.

Once the classes are created, the next step is to create the properties that relate the different classes to each other and are used to define the features. This is handled in the Properties tab of Protégé-OWL, where the properties, subproperties, the domains and ranges of the properties are defined. For an object property, it can be assigned with the options of function, inverse function, symmetric, or transitive. For a datatype property, it can only be a function instead of an inverse function, nor can it be symmetric or transitive. See Appendix A of Ref. [45] for the complete set of the shared base ontology.

4.2. Feature data export from source system

We use Pro/Engineer as the source system to demonstrate. The example in Fig. 12 is used to illustrate. Source feature data are

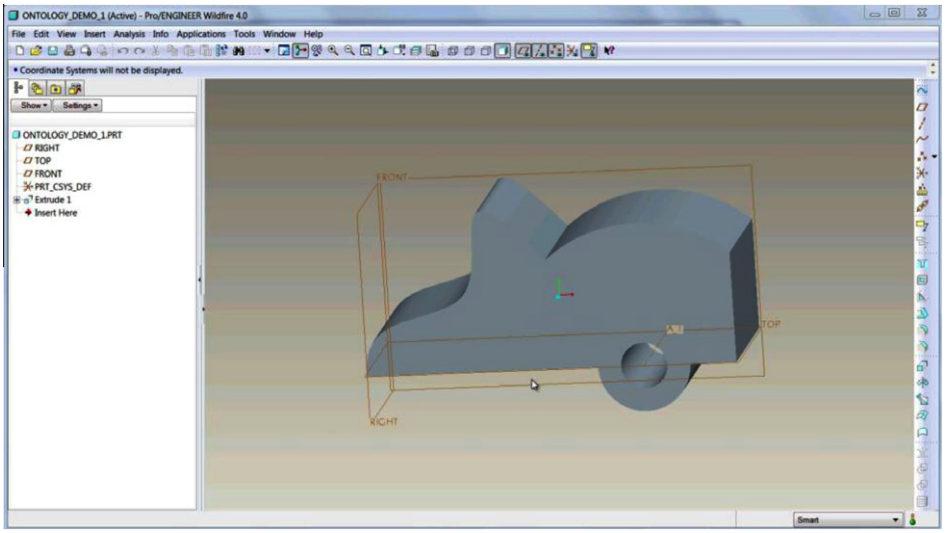


Fig. 12. An example model built in Pro/Engineer.

extracted from part models in Pro/Engineer through the use of its API, Pro/TOOLKIT. Pro/TOOLKIT provides functions to automatically export each feature in the construction history tree as an XML file, which however does not contain 2D sketch data. A C++ program was thus developed to export 2D sketch data including entities, constraints, and dimensions from Pro/Engineer to a file in the shared base ontology OWL format, shown in Fig. 13. A separate code was also developed to create a log file of each feature operation and to export the B-Rep data in ACIS format. The ACIS format was chosen because of its fairly straightforward data structure. A third program reads in the XML feature files, the OWL sketch files, and the ACIS B-Rep files and appends them all into a single global CAD ontology part file stored in the OWL format. A sample section of the fully formatted OWL file is presented in Fig. 14. This OWL file has an ontology import tag recognizable by Protégé.

The reading, parsing, converting, and exporting process need to be done separately for each feature. Only the extrude feature and the features that define the reference datum and coordinate system are used in this example. Other feature types can be done similarly. Once the feature construction history has been appended to the OWL file, the program checks the 2D sketches and adds them to the appropriate extrude feature based on the given reference. Finally the B-Rep data are read in, line by line, with each ACIS class type examined. The geometric information is appended to the OWL file as a series of B-Rep entity creations. The complete B-Rep data can be examined from within Protégé-OWL, as shown in Fig. 15. But only a generic *Feature* class is recognized. Again, the issue of persistent naming for the B-Rep model is ignored here as it is out of the scope of this work. A simple scheme of numerical identifiers is used in the implementation.

4.3. Local ontology feature definitions and classification with SWRL rules

Once the shared CAD information in the OWL format is exported from the source system, it can be automatically mapped

```
<CAD:Sketch rdf:ID="S2D0003">
  <CAD:hasCoordAxisEntity2D>
    <CAD:CoordAxisEntity2D rdf:ID="Section_S2D0003_Entity_0">
      <CAD:hasStartPoint>
        <CAD:PointEntity2D rdf:ID="Section_S2D0003_Entity_0_StartPoint">
          <CAD:hasXCoord rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.00</CAD:hasXCoord>
          <CAD:hasYCoord rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.00</CAD:hasYCoord>
        </CAD:PointEntity2D>
      </CAD:hasStartPoint>
      <CAD:hasEndPoint>
        <CAD:PointEntity2D rdf:ID="Section_S2D0003_Entity_0_EndPoint">
          <CAD:hasXCoord rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.00</CAD:hasXCoord>
          <CAD:hasYCoord rdf:datatype="http://www.w3.org/2001/XMLSchema#float">-100.00</CAD:hasYCoord>
        </CAD:PointEntity2D>
      </CAD:hasEndPoint>
    </CAD:CoordAxisEntity2D>
  </CAD:hasCoordAxisEntity2D>
  <CAD:hasCoordAxisEntity2D>
    <CAD:CoordAxisEntity2D rdf:ID="Section_S2D0003_Entity_1">
      <CAD:hasStartPoint>
        <CAD:PointEntity2D rdf:ID="Section_S2D0003_Entity_1_StartPoint">
          <CAD:hasXCoord rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.00</CAD:hasXCoord>
          <CAD:hasYCoord rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.00</CAD:hasYCoord>
        </CAD:PointEntity2D>
      </CAD:hasStartPoint>
      <CAD:hasEndPoint>
        <CAD:PointEntity2D rdf:ID="Section_S2D0003_Entity_1_EndPoint">
          <CAD:hasXCoord rdf:datatype="http://www.w3.org/2001/XMLSchema#float">100.00</CAD:hasXCoord>
          <CAD:hasYCoord rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.00</CAD:hasYCoord>
        </CAD:PointEntity2D>
      </CAD:hasEndPoint>
```

Fig. 13. A sample section of OWL sketch file.

```
<CAD:hasFeature>
    <CAD:Feature rdf:ID="EXTRUDE_1">
        <CAD:hasReferenceSketch>
            <CAD:Sketch rdf:ID="S2D0002">
        </CAD:hasReferenceSketch>
        <CAD:hasExtSurfaceType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Solid</CAD:hasExtSur
        <CAD:removesMaterial rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">False</CAD:removesMat
        <CAD:isThinShellPart rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">False</CAD:isThinShel
        <CAD:hasDepthDimension rdf:datatype="http://www.w3.org/2001/XMLSchema#float">100.000000</CAD:hasDe
        <CAD:hasPrimaryReferenceDatumPlane rdf:resource="#FRONT" />
        <CAD:hasSecondaryReferenceDatumPlane rdf:resource="#RIGHT" />
        <CAD:hasNumberReferenceSketchInstances rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</CAD:
        <CAD:createsShell>
            <CAD:Shell rdf:ID="BRepID-3" />
        </CAD:createsShell>
        <CAD:createsFace>
            <CAD:Face rdf:ID="BRepID-4">
                <CAD:hasLoop rdf:resource="#BRepID-6" />
                <CAD:isFaceOf rdf:resource="#BRepID-3" />
                <CAD:hasGeometryOfSurface rdf:resource="#BRepID-7" />
            </CAD:Face>
        </CAD:createsFace>
        <CAD:createsFace>
            <CAD:Face rdf:ID="BRepID-5">
                <CAD:hasLoop rdf:resource="#BRepID-9" />
                <CAD:isFaceOf rdf:resource="#BRepID-3" />
                <CAD:hasGeometryOfSurface rdf:resource="#BRepID-10" />
            </CAD:Face>
        </CAD:createsFace>
```

Fig. 14. A sample section of OWL exchange file as shared base ontology.

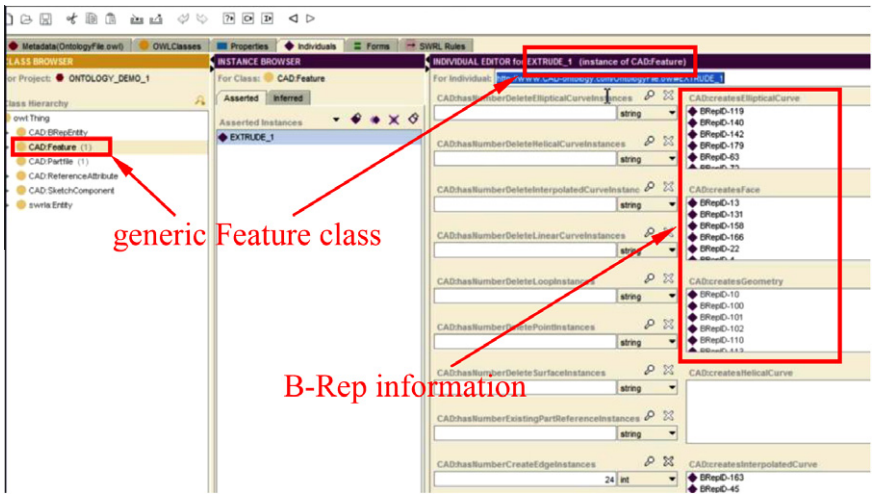


Fig. 15. Protégé-OWL recognizes the shared OWL file as a generic *Feature*.

to local systems according to the local feature definitions and rules. A local feature hierarchy was created similar to the one in Fig. 9. SWRL rules to classify *SketchBasedFeature*, *EdgeFeature*, and *ExtrudeFeature* were implemented. Besides rules to define local features, additional rules were created to compute the inverse properties of some of the B-Rep properties. This was necessary because when instances are imported from an external file, inverse properties are not automatically created unless they were previously defined in the imported file. The ontology reasoners built into Protégé-OWL do not automatically infer the inverse properties either, which means that the rules have to be added to ensure some of the B-Rep entities are fully defined.

When the shared OWL part file is imported in the local CAD ontology file in Protégé-OWL, the feature has not been classified and it is merely an instance of the *Feature* parent class. To classify it, the SWRL rules must be run using the Jess rule engine. The test to check if a feature is an instance of the *SketchBasedFeature* class is to see if the feature has one reference sketch associated with it. If this is true, then the feature becomes an instance of the *SketchBasedFeature* class. The test to determine an instance of the *EdgeFeature* class is similarly simple. This rule checks if all reference attributes are of type *ReferenceEdge*. Proceeding downwards in the class hierarchy, the test to determine if an individual of the *SketchBasedFeature* class is a member of the *ExtrudeFeature* class is slightly more complex. This test compares the number of 2D sketch entities to the number of specific surface types with only B-Rep creation operations, and the exact number of surface creations could be predicted. The test for the *ExtrudeFeature* in SWRL format is displayed in Fig. 16, where the variable 'F' stores instances of *SketchBasedFeature* class, and 'S' stores the instances of the sketches for each feature in Line 1. Line 2 to Line 4 checks if the number of planar surfaces equals the number of line entities plus two. Line 4 to Line 8 sums the number of circle and arc entities and ensures that there is a conical surface for each one. Line 9 to

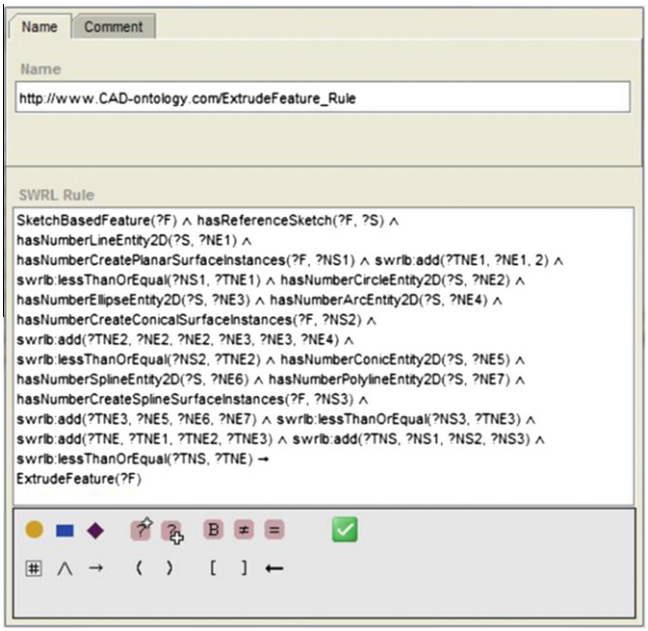


Fig. 16. SWRL rule for *ExtrudeFeature* classification.

Line 11 tests if the total number of spline surfaces is equal to the total number of 2D ellipse, conic, spline, and polyline entities. Finally, Line 12 to Line 13 sums all the sketch entities and the number of surfaces created, and then checks if the number of surfaces is equal to the number of sketch entities plus two. If all of the above is true, then Line 14 assigns the object stored in 'F' as a member of the *ExtrudeFeature* class.

Note that the rules are nothing more than a series of AND logical conjunctions. If any part of the rule fails, the inference will not apply. If the feature were interacting with an existing model, these rules would no longer hold true. Such rules would have to include exceptions, and equivalent tests that incorporate the feature creation and validation rules need to be employed. After the rule in Fig. 16 is applied, the extrude feature in the example can be recognized by Protégé-OWL as shown in Fig. 17.

Fig. 18 shows a SWRL rule to classify a feature as a member of the *RevolveFeature* class. It ensures that the sum of planar and conical surfaces created by 'F' must be less than or equal to the sum of line entities belonging to 'S'; the sum of toroidal and

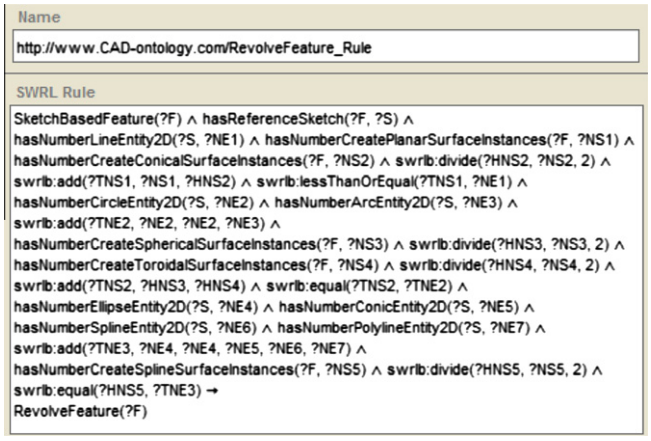


Fig. 18. SWRL rule for *RevolveFeature* classification.

spherical surfaces belonging to 'F' must equal the number of circle and arc entities belonging to 'S'; and the number of spline surface belonging to 'F' must equal the sum of ellipse, conic, polyline, and spline entities belonging to 'S'. This rule is only true when the angle of rotation is equal to 360°. When it is less, two additional planar surfaces have to be recognized, in which case a different rule has to be created since OR operators cannot be used.

The above examples illustrate that SWRL rules can be used to classify features without relying solely on semantics of feature definitions. Geometry information can significantly reduce the ambiguity in the feature mapping process. Yet, defining sufficient and necessary rules for accurate and robust classification requires in-depth knowledge of CAD systems and the complexity of the rules will increase. Rule exceptions will occur if features are used to add or remove materials to an existing model. Because with only AND operators, more rules would be needed for any exception. In addition, without checking if the parameters of the surfaces correspond to the entities in the sketch, it could be possible to generate false positives.

4.4. Limitations of SWRL rules

SWRL rules were built to infer new property relationships between existing individuals. However, SWRL shares OWL's *open world* assumption, which restricts some reasoning abilities. In the open world assumption, something cannot be determined to not

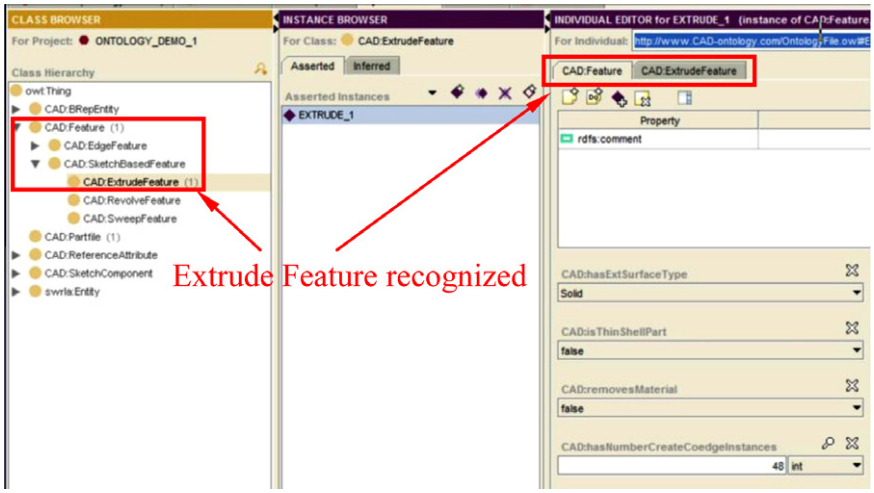


Fig. 17. Protégé recognizes the extrude feature after the *ExtrudeFeature* SWRL rule is applied.

exist until explicitly stated. For example, if there is a class *Parent* with a restriction that any individual with property *hasChild* is a *Parent*, then only conclusions can be made about individuals that have children. If the individual does not have any instances of the *hasChild* property, it cannot be concluded that the individual is not a *Parent*. The open world assumption works this way to allow for introduction of new data over time. This prevents new information from invalidating previous conclusions. Unfortunately, this has some major implications. SWRL only supports *monotonic* inferencing, so SWRL rules cannot modify or remove existing information from the ontology. The monotonicity also means SWRL rules cannot support negation (NOT operator), nor disjunction (OR). Only conjunction (AND) statements can be used. Nevertheless, it is still possible to work around these limitations. Negation can be expressed if there is a property explicitly stating an individual cannot be a member of a class, or if an integer datatype property is used to specify that there are no instances of specific property. However, the open world assumption greatly increases the difficulty of expressing any feature recognition rule in a robust manner. Closed world reasoning can be made possible by introducing integer datatype properties to count specific instances of a class. Yet, these integers must be instantiated and exported by the program that creates the OWL exchange file.

In OWL, object properties relate an individual of one class to individuals of another class with cardinalities specified as the set of necessary and sufficient property restrictions placed upon them. The cardinality restriction can be specified as greater than, less than, or exactly a fixed integer number. For example, an extrude feature could have the property restriction 'Cardinality(*hasSketch*) = 1'. However, the cardinality cannot be specified as a variable or equation. Rules such as 'Cardinality(*createsPlaneSurface*) $\leqslant$ Cardinality(*hasLineEntity*)' cannot be used. Property restrictions on the values of datatype properties are not allowed either. Because the restrictions available to OWL classes are limited, we had to use combinations of rules, which could have been simpler if the cardinality restriction were flexible enough.

5. Concluding remarks

While ontology has been regarded as a powerful tool to solve interoperability for heterogeneous data, the existing approaches for CAD data exchange rely too heavily on standard neutral format with rigid feature specifications or direct mapping purely based on semantic similarity of class-level feature definitions. Approaches that do not emphasize automation will likely be too source consuming to develop and maintain for affordable and effective exchange. Our solution presented in this paper emphasizes a CAD feature ontology should be based on well established, understandable, and fundamental nomenclature while retaining necessary freedom to model the wide variety of features available in different CAD systems. The proposed approach focuses on describing features in a more fundamental way via a shared global base ontology, so as to allow for improved automation in the mapping process. By introducing SWRL rules to serve as necessary and sufficient conditions for defining features locally in different CAD systems, features in the base form with universally accepted information elements can be recognized as members of a given class in the target system before any semantic mapping process occurs, thereby eliminating unnecessary semantic mapping procedures. With such an approach, features that represent the same shape concept but are semantically or structurally different could still be identified as matches.

It should be noted that the main strength of this approach is its capability to support exchange in an open and distributed environment with no predetermined feature standards. Each CAD system would only be responsible for exporting data in the shared base ontology format which are no more than the basic geometry, topology, sketch dimensions and constraints, and for creating rules to import data from this format. No specific knowledge of the other systems is required. In contrast, existing commercial one-to-one translators require extensive knowledge of the target systems. With the assumption of such prior knowledge, ad hoc mapping procedures for a few of the extensively used systems could be very efficient. Yet our approach is meant to demonstrate the foundation on which an ontology-based neutral exchange format could be developed for its intended flexibility, which also eliminates unnecessary semantic similarity calculations by rule-based reasoning. With future improvement of property restrictions in OWL, such rules could be incorporated into the class definitions more formally.

The main benefits that the proposed approach will bring to CAD software vendors are summarized as follows. The dynamic mapping approach does not require the library of the source system during the data exchange process, therefore companies' proprietary feature information is protected without the need of sharing. When a new feature or a new CAD system is introduced, it does not affect the standardized shared base ontology which only contains the fundamental elements of the widely accepted 2D sketch and B-Rep information. Therefore, CAD vendors have the freedom to introduce new features without incurring new issues of interoperability. In addition, by adopting the new ontology format to formally document the feature definitions and the existing rules for feature evaluation and verification used in their CAD software systems, companies can have a systematic way to manage corporate internal knowledge with ease of retention, retrieval, and reuse.

Much work still needs to be done in developing improved mapping processes for heterogeneous CAD data using ontologies. Representing a feature in terms of necessary and sufficient conditions based on geometry creation rules needs more efforts to capitalize its full benefits over a purely semantic approach. As incorporating the closed world reasoning into OWL is an ongoing effort by researchers (e.g. Grimm et al. [46,47], Katz and Parsia [48], and Knorr et al. [49]), the necessary and sufficient property restriction rules could possibly be extended to allow for full description of feature concepts. The current open world assumption reasoning of OWL only benefits building databases with inserting new information. When using an ontology as an exchange format, all information that will be used is included in the file, so there is no addition of new data that could invalidate previous logic determinations.

Other areas of future work include methods to improve the extraction and storage of feature data. An improved way, with the persistent naming issue resolved, to store B-Rep data in the ontological format after each feature creation in the history tree is needed. Complete local feature ontologies for different CAD systems are also desirable. However, the creation of such local ontologies will require a deep understanding of each target system's feature library and how each feature is defined and geometrically constructed. Further deliberation of better matching beyond B-rep data is needed when there is no single feature in the target system that adequately resembles the shape described in the base ontology. Any level of automation for feature composition is noteworthy. Once the complete translation of features between all available CAD systems (both commercial and open-source) is achieved, more research needs to be done to comprehensively compare the efficacy of the new approach versus a purely semantic approach, or a purely ad hoc one-to-one approach. As another benefit of this approach is its support of instance-level dynamic mapping processes, future research could be done in a completely distributed environment with research groups focusing on individual CAD systems. This would enable research on collaborative

design to progress more quickly without the need of expertise on multiple CAD systems to contribute.

Acknowledgments

This work is supported by the National Science Foundation Industry-University Cooperative Research Center (I/UCRC) for e-Design. We would like to thank John Altidor and Professor Jack Wileden for their discussions and feedback, as well as the Pro/Engineer XML export software code they shared with us.

References

[1] TTI, Acc-u-Trans 6.0. <http://www.translationtech.com/accutrans_6_0.asp>.
[2] TranscenData, Proficiency. <http://www.transcendata.com/products/proficiency/>.
[3] W.F. Bronsvoort, F.W. Jansen, Feature modelling and conversion—key concepts to concurrent engineering, Computers in Industry 21 (1993) 61–86.
[4] J.J. Shah, D. Anderson, Y.S. Kim, S. Joshi, A discourse on geometric feature recognition from CAD models, Journal of Computing and Information Science in Engineering 1 (2001) 41.
[5] W.C. Regli, S.K. Gupta, D.S. Nau, Extracting alternative machining features: an algorithmic approach, Research in Engineering Design 7 (1995) 173–192.
[6] J. Han, A.A.G. Requicha, Integration of feature based design and feature recognition, Computer-Aided Design 29 (1997) 393–403.
[7] X. Chen, C.M. Hoffmann, On editability of feature-based design, Computer-Aided Design 27 (1995) 905–914.
[8] C.M. Hoffmann, EREP project overview, in: CAD Systems Development: Tools and Methods [Dagstuhl Seminar, 1995], Springer-Verlag, 1997, pp. 32–40.
[9] G.-H. Choi, D. Mun, S. Han, Exchange of CAD part models based on the macro-parametric approach, International Journal of CAD/CAM 2 (2002) 13–21.
[10] D. Mun, S. Han, J. Kim, Y. Oh, A set of standard modeling commands for the history-based parametric approach, Computer-Aided Design 35 (2003) 1171–1179.
[11] T.-S. Seo, Y. Lee, S.-U. Cheon, S. Han, L. Patil, D. Dutta, Sharing CAD models based on feature ontology of commands history, International Journal of CAD/CAM 5 (2005) 39–48.
[12] I. Song, S. Han, Parametric CAD data exchange using geometry-based neutral macro file, Lecture Notes in Computer Science 6240 (2010) 145–152.
[13] V. Capoyleas, X. Chen, C.M. Hoffmann, Generic naming in generative, constraint-based design, Computer-Aided Design 28 (1996) 17–26.
[14] J. Kripac, A mechanism for persistently naming topological entities in history-based parametric solid models, Computer-Aided Design 29 (1997) 113–122.
[15] D. Marcheix, G. Pierra, A survey of the persistent naming problem, in: Proc. the Seventh ACM Symposium on Solid Modeling and Applications ACM, Saarbrücken, Germany, 2002, pp. 13–22.
[16] Y. Wang, B.O. Nnaji, Geometry-based semantic ID for persistent and interoperable reference in feature-based parametric modeling, Computer-Aided Design 37 (2005) 1081–1093.
[17] R. Bidarra, P.J. Nyirenda, W.F. Bronsvoort, A feature-based solution to the persistent naming problem, Computer-Aided Design and Applications 2 (2005) 517–526.
[18] A. Rappoport, An architecture for universal CAD data exchange, in: Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications, ACM, Seattle, Washington, USA, 2003, pp. 266–269.
[19] A. Rappoport, S. Spitz, M. Etzion, Two-dimensional selections for feature-based data exchange, Lecture Notes in Computer Science 4077 (2006) 325–342.
[20] S. Spitz, A. Rappoport, Integrated feature-based and geometric CAD data exchange, in: Proceedings of the Ninth ACM Symposium on Solid Modeling and Applications, Eurographics Association, Genoa, Italy, 2004, pp. 183–190.
[21] M. Li, S. Gao, C.C.L. Wang, Real-time collaborative design with heterogeneous CAD systems based on neutral modeling commands, Journal of Computing and Information Science in Engineering 7 (2007) 113–125.
[22] C.-H. Shih, B. Anderson, A design/constraint model to capture design intent, in: Proceedings of the Fourth ACM Symposium on Solid Modeling and Applications, ACM, Atlanta, Georgia, USA, 1997, pp. 255–264.
[23] X. Cai, F. He, X. Li, X. Chen, A direct feature retrieval and reuse method for heterogeneous collaborative CAD systems (2008) 718–723.
[24] X. Li, F. He, X. Cai, D. Zhang, CAD data exchange based on the recovery of feature modelling procedure, International Journal of Computer Integrated Manufacturing (2011) 1–14.
[25] Y. Zhang, X. Luo, Design intent information exchange of feature-based CAD models (2009) 11–15.
[26] W. Sun, T.-Q. Ma, Y.-J. Huang, Research on method of constraint conversion in feature-based data exchange between heterogeneous CAD systems, Journal of Mechanical Science and Technology 23 (2009) 246–253.
[27] J. Kim, M.J. Pratt, R.G. Iyer, R.D. Sriram, Standardized data exchange of CAD models with design intent, Computer-Aided Design 40 (2008) 760–777.
[28] L. Patil, D. Dutta, R. Sriram, Ontology-based exchange of product data semantics, IEEE Transactions on Automation Science and Engineering 2 (2005) 213–225.
[29] Y. Wang, B.O. Nnaji, Document-driven design for distributed CAD services in service-oriented architecture, Journal of Computing and Information Science in Engineering 6 (2006) 127–138.
[30] C. Dartigues, P. Ghodous, M. Gruninger, D. Pallez, R. Sriram, CAD/CAPP integration using feature ontology, Concurrent Engineering 15 (2007) 237–249.
[31] O. Kim, U. Jayaram, S. Jayaram, L. Zhu, An ontology mapping application using a shared ontology approach and a bridge ontology, in: Proceedings of ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2009), ASME, San Diego, California, USA, 2009, pp. 431–441.
[32] P. Zhan, U. Jayaram, O. Kim, L. Zhu, Knowledge representation and ontology mapping methods for product data in engineering applications, Journal of Computing and Information Science in Engineering 10 (2010) 021004.
[33] L. Hanayneh, Y. Wang, Y. Wang, J.C. Wileden, K.A. Qureshi, Feature mapping automation for CAD data exchange, in: Proceedings of ASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2008), ASME, Brooklyn, New York, USA, 2008, pp. 1257–1266.
[34] J. Altidor, J. Wileden, Y. Wang, L. Hanayneh, Y. Wang, Analyzing and implementing a feature mapping approach to CAD system interoperability, in: Proceedings of ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2009) San Diego, California, USA, 2009, pp. 695–707.
[35] M.R. Henderson, D.C. Anderson, Computer recognition and extraction of form features: a CAD/CAM link, Computers in Industry 5 (1984) 329–339.
[36] S. Prabhakar, M.R. Henderson, Automatic form-feature recognition using neural-network-based techniques on boundary representations of solid models, Computer-Aided Design 24 (1992) 381–393.
[37] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, S. Hübner, Ontology-based integration of information – a survey of existing approaches, in: IJCAI-01 Workshop on Ontologies and Information Sharing, Seattle, WA, 2001, pp. 108–117.
[38] M. Baba-ali, D. Marcheix, X. Skapin, A method to improve matching process by shape characteristics in parametric systems, Computer-Aided Design and Applications 6 (2009) 341–350.
[39] OWL_Working_Group, OWL Web Ontology Language – overview. W3C, 2003. <http://www.w3.org/TR/owl-features/>.
[40] RDF_Working_Group, Resource Description Framework (RDF), 2004. <http://www.w3.org/RDF>.
[41] I. Horrocks, P.F. Patel-Schneider, F. Van Harmelen, From SHIQ and RDF to OWL: the making of a web ontology language, Web Semantics: Science, Services and Agents on the World Wide Web 1 (2003) 7–26.
[42] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A semantic web rule language combining OWL and RuleML, W3C, 2004. <http://www.w3.org/Submission/SWRL>.
[43] H. Knublauch, R.W. Fergerson, N.F. Noy, M.A. Musen, The Protégé OWL Plugin: an open development environment for semantic web applications, Lecture Notes in Computer Science 3298 (2004) 229–243.
[44] E. Friedman-Hill, Jess – The Rule Engine for the Java Platform, Sandia National Laboratories, 2003.
[45] S.M. Tessier, Ontology-based approach to enable feature interoperability betwen CAD systems, in: Mechanical Engineering, Georgia Institute of Technology, Atlanta, Georgia, 2011.
[46] S. Grimm, B. Motik, C. Preist, Matching semantic service descriptions with local closed-world reasoning, Lecture Notes in Computer Science 4011 (2006) 575–589.
[47] P. Hitzler, S. Grimm, Semantic matchmaking of web resources with local closed-world reasoning, International Journal of Electronic Commerce 12 (2007) 89–126.
[48] Y. Katz, B. Parsia, Towards a nonmonotonic extension to OWL, in: Proceedings of Workshop on OWL Experiences and Directions, Galway, Ireland, 2005.
[49] M. Knorr, J.J. Alferes, P. Hitzler, Local closed world reasoning with description logics under the well-founded semantics, Artificial Intelligence 175 (2011) 1528–1554.