Proceedings of the ASME 2009 International Design Engineering Technical Conferences &
Computers and Information in Engineering Conference
IDETC/CIE 2009
August 30-September 2, 2009, San Diego, USA

DETC2009-87539

ANALYZING AND IMPLEMENTING A FEATURE MAPPING APPROACH TO CAD SYSTEM INTEROPERABILITY

John Altidor
Jack Wileden

Department of Computer Science
University of Massachusetts
Amherst, MA 01003
Email: {jaltidor,wileden}@cs.umass.edu

Yiwen Wang
Leen Hanayneh
Yan Wang

Department of Industrial Engineering and Management Systems
University of Central Florida
Orlando, FL 32816
Email: wienwang@gmail.com
leen.hanayneh@gmail.com
wangyan@mail.ucf.edu

ABSTRACT

Interoperable information exchange between computer-aided design (CAD) systems is one of the major problems to enable information integration in a collaborative engineering environment. Although a significant amount of work has been done on the extension and standardization of CAD data formats as well as the cooperation of CAD systems in both academy and industry, these approaches are generally low-level and narrowly targeted. Lack of fundamental study of interoperability and generic solution to this problem is the major issue. Our intention of this research is to design a solution of CAD feature interoperability as generic as possible based on a theoretical foundation of language types. In this paper, we present a fundamental model of semantic features and feature mapping process based on the type theory. We implement and demonstrate our approach for automated feature exchange between commercial CAD systems.

1 INTRODUCTION

One of the major cost elements associated with product design is the precious time engineers spend on data translation between different computer aided design (CAD) formats, data integration between different systems, routine data reprocessing in different application scenarios, redesign due to loss of information, and error correction due to human errors in the above processes. Complementary to data exchange standards such as the Standard for the Exchange of Product Model Data (STEP), design process automation can reduce time and cost in a virtual collaborative engineering environment. Interoperable CAD model generation, intent and knowledge capturing, and semantic-level information exchange are critical to enable the automation. Interoperability is a multi-level problem. At the low level, the concerns involve simply getting tools to communicate and getting data exchanged without corruption. At the mid level, semantic information embedded in different domain syntax structures can be interpreted losslessly. At the high level, domain knowledge, ontology, and philosophical meta-knowledge are transferable and propagated across disciplinary boundaries. These are, in reality, nontrivial concerns, especially when the tools are heterogeneous and the data are complex.

Software companies usually take a static identify-and-map approach to improve feature-level interoperability between CAD systems by understanding and using the available application programming interfaces (APIs) from the systems, which are generally one-to-one, ad-hoc, low-level, narrowly targeted, and prone to produce erroneous interactions. The majority of the

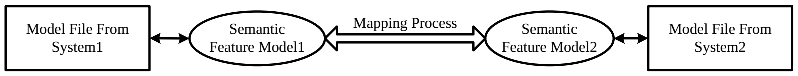


Figure 1. STRUCTURE OF DATA EXCHANGE AUTOMATION APPROACH

work by academic researchers focuses on interoperable feature modeling. Our objective is to design a solution of CAD feature interoperability as generic as possible based on a theoretical foundation of language types.

In this paper, a fundamental model of semantic features and feature mapping process based on the type theory are proposed. Our approach enables automating the data exchange between different 3D CAD systems as illustrated in Figure 1. This approach provides a neutral semantic feature format. The semantic feature files can be created by extracting the information from CAD models through a system's API. The semantic feature definitions are modeled by graph structures and represented as resource description framework (RDF) files. We further develop a two-step feature mapping process to transfer features from one system to another. In the first step, semantic feature files are preprocessed by a parser subjected to type checking. In the second step semantic features matching is accomplished based on attribute subgraph similarity. This mapping process is not restricted to one-to-one mapping. It can be applied to many-to-many mappings to reduce the complexity of collaboration among many systems. This new approach is built on both computer science and engineering principles to improve the interoperability at both language and data semantics levels.

The advantages of our approach are listed as follows.

1. Compared to the static identify-and-map approach, our approach reduces the complexity of searching and mapping processes for multiple systems. The time used in this process has a linear relationship with the number of systems whereas the traditional one-by-one translation process has a factorial complexity and is difficult to manage as the number of systems grows [1].
2. A multi-resolution approach is taken in the hybrid semantic feature modeling to record the design information. Model construction information can be abstracted at multiple levels. This approach is to support extensible feature modeling. When system-specific feature definitions are changed at the low-level, the high-level model is not affected [1].
3. The similarity-based mapping algorithm enables real-time feature model matching. Without prior definitions, new feature data can be mapped in an incremental way. A document-driven design (DDD) mechanism [2] can be taken to automate the process of feature generation. The DDD mechanism is developed to support history-neutral modeling in CAD systems, by which the construction of features is independent of design history. Every feature can exist in a separate document. Creating a new feature or modifying an existing feature can be achieved by only adding a new document or modifying the relevant document.
4. The DDD mechanism also increases the flexibility of model exchange. Separate documents can be used to capture individual features. As a result, a fat-server-thin-client approach can be adopted in a distributed design environment. The server may be used to keep the document repository, preview 3D models in neutral feature format, and translate between the neutral file format and the native format. APIs can be installed on the client side to capture specific features [2].

In previous work [1], we proposed an automated feature mapping approach to find the correspondence between features in two CAD systems based on subgraph isomorphism. In this paper, we define a formal model of semantic features and feature mapping process. We also demonstrate *static* and *dynamic* procedures for feature mapping automation. In the static mapping procedure, class-level feature definitions between CAD systems are matched to generate translators. For dynamic mapping, instance-level features are used. This is useful when new features or new CAD systems are introduced and no library is available. Dynamic mapping can also be useful in data synchronization in collaborative engineering.

In the remainder of the paper, Section 2 gives a brief review of recent research on feature modeling for interoperable data exchange. Graph matching algorithms and programming language formalisms for type checking and formal semantics are also introduced. Section 3 describes our hybrid semantic feature model, develops a formal description for it, and describes its realization in RDF. Section 4 presents the feature mapping algorithm to enable static and dynamic mapping processes. In section 5 we formally demonstrate several important properties of the feature mapping approach. Section 6 implements and demonstrates our approach in commercial CAD systems.

2 BACKGROUND

2.1 3D CAD Data Exchange

Features are widely used in modern CAD systems as the basic units of geometry construction. Lack of standard definitions of features causes the issue of interoperability. There are plenty of research efforts on 3D CAD data exchange. The widely used standards are IGES and STEP which use the boundary representation (B-Rep) to represent the geometry information of a model. The drawback of this pure B-Rep approach is that it can only treat a model as a uniform solid shape without the information of construction history and design intent.

Based on current framework of STEP standard, a number of researchers have tried to extend STEP to include more information from the model construction process. E-Rep [3,4] distinguishes between generated features, datum features, and modifying features; it regards a CAD model as being built entirely by a sequence of feature insertion, modification, and deletion description. The ENGEN Data Model (EDM) [5] extended STEP's current explicit entity representation by adding some predefined local features such as round and chamfer in a bottom-up approach. PDES's Form Feature Information Model (FFIM) [6, 7] adopted a dual representation of explicit and implicit features. Explicit features are represented generally by face lists, while implicit features are categorized into depression, protrusion, passage, deformation, transition, and area features. Kim et al [8] define features, dimensions, constraints and construction history information on top of the STEP standard. Their approach focuses on the intersystem exchange of construction history with parameterization and constraints. Different from the above STEP-based approaches, Rappoport et al. [9] introduced a representation of features by the B-Rep structure. They proposed a concept of "feature rewrite" which describes the geometry difference before and after adding a feature to the original model. The interoperability is achieved by transforming geometry per feature (GPF) among systems.

A different approach is to translate the history of model creation instead of the model itself from one system to another. In this approach, APIs of the source and target systems were used to record design history by system modeling operations. Choi et al. [10] proposed a macro parametric approach to provide capabilities to exchange parametric information. This approach is to translate the macro recorded in the source system to a neutral macro format which contains only the actions to create the model, and then translate it to the macro of the target system. Li et al. [11] established a real-time collaborative design environment based on the translation between system modeling operations (SMOs) with neutral modeling commands (NMCs). In this system, the collaboration can be fulfilled in real time with different systems and multiple clients synchronously.

Different from the above methods, we are interested in improving interoperability by looking into feature semantics in a more general way across the boundary of data and programming languages, as interoperability between programming languages is also an important issue in computer science. We assimilate CAD features as language types and propose a systematic approach to map feature semantics based on type checking.

2.2 Formal Semantics and Type Theory

Type theory [12] is an abstract formalism used extensively in the study and design of programming languages to define the semantics and behavior of automated systems. Ridgway and Wileden have presented a type system of interoperability based on extensions to the resource-bounded effects approach; the static and dynamic semantics can be used as a basis for establishing whether programs in two different programming languages will or will not interoperate correctly relative to some higher-level properties [13, 14]. Matthews and Findler developed a type system to study semantic properties, such as observable equivalence, of multi-language programs [15]. Simeon and Wadler have convincingly demonstrated the power of type systems as a basis for analyzing, reasoning about, and improving the XML data exchange standard [16].

CAD system interoperability suffers from vulnerabilities [17] that can be addressed with type systems that formally model interoperability and automatically check for safety properties for integration. Section 5 describes our approach to type-checking CAD model feature mappings, which enables a system to check whether converting one model by substituting one feature for another would result in another model that could be used as a replacement to the original model. The remainder of this section will provide a brief tutorial on type systems and the notation used in their formalisms.

Type systems are a set of *judgments* or assertions about an object of study. Judgments are defined by *inductive definitions*, which consists of *inference rules* of the form:

$$\frac{J_1 \quad J_2 \quad \dots \quad J_k}{J}$$

where $J_1, J_2, \dots, J_k, J$ are all judgments of similar form. Judgments above the horizontal line are the premises of the rule, and below the line is the conclusion of the rule. If no premises were above the line, then the rule would be called an *axiom* since the conclusion is true without any pre-conditions.

For example, the following two rules inductively define the set of natural numbers:

$$\frac{}{\texttt{zero : nat}}\ \text{NAT\_ZERO} \qquad \frac{\texttt{a : nat}}{\texttt{succ(a) : nat}}\ \text{NAT\_SUCC}$$

The notation `a : nat` asserts that the object `a` has type `nat`; in other words, `a` is a natural number. Rule `NAT_ZERO` is an axiom stating that `zero` is a natural number. `succ(a)` can be thought of as the function `succ(a) = a+1`. Rule `NAT_SUCC` states that if `a` is a natural number, then `succ(a)` is also a natural number.

To check if an object `a` has type *nat*, a system can search for a *derivation* of the judgment `a : nat`. A derivation of a judgment is a composition of rules, starting with axioms and ending with that judgment. Derivations are represented as proof trees in which each node is a rule whose children are derivations of its premises. The example proof tree below derives the judgment `succ(succ(succ(zero))) : nat`:

$$\frac{\frac{\frac{\frac{}{\texttt{zero : nat}}}{\texttt{succ(zero) : nat}}}{\texttt{succ(succ(zero)) : nat}}}{\texttt{succ(succ(succ(zero))) : nat}}$$

The notation for typing judgments such as `a : nat` is slightly changed to an infix form when defining relations between two objects. For instance, to define equality, instead of using notation such as `(a, b) : =` to say that the ordered pair `(a, b)` has type $=$, we use the more natural infix notation of $a = b$. The example two rules below define the equality relation over natural numbers.

$$\frac{}{\texttt{zero = zero}}\ \text{EQ\_ZERO} \qquad \frac{\texttt{a = b}}{\texttt{succ(a) = succ(b)}}\ \text{EQ\_SUCC}$$

Lastly, by the *principle of rule induction*, to show that a property P holds for a judgment, it is enough to show that for every rule concluding that judgment:

$$\frac{J_1 \quad J_2 \quad \dots \quad J_k}{J}$$

if $J_1, J_2, \dots, J_k$ each have property P, then J has property P. In other words, for each rule of the judgment, show that if each premise has property P, then the conclusion has property P. For example, to prove that every `nat` is nonnegative:

1. For rule NAT_ZERO, show that `zero` is nonnegative.
2. For rule NAT_SUCC, assuming that `a : nat` and `a` is nonnegative, show that `succ(a)` is nonnegative.

Since there are no other derivations of the judgment `a : nat`, then the property of nonnegativeness must hold for all `nat`'s.

2.3 Graph Matching

Our underlying mechanism for feature mapping is graph matching. A graph $G = (V, E)$ in its basic form is composed of vertices V and edges E. V is the set of vertices and $E \subseteq V \times V$ is the set of edges in graph G. The order of a graph G is defined as the number of vertices of G and denoted $|V|$ and the number of edges as $|E|$. Vertices and edges can also be labeled providing more information about a graph. A graph with such information is called a labeled graph. Moreover, the vertices and edges can have attributes giving an attributed graph. Graph matching involves matching a graph with another graph or subgraph by mapping vertices and edges. When two graphs are structurally identical, i.e. every node and edge in one graph is only mapped once, they are said to be isomorphic. When two graphs are not isomorphs, the goal will be in finding the most similar subgraphs.

Many algorithms have been developed to check graph isomorphism such as the Ullmann algorithm [18], NAUTY algorithm [19], VF algorithm [20], and the Schmidt and Druffel algorithm [21]. All the above algorithms are deterministic. In contrast, the algorithm developed by Corneil and Gottlieb [22] is non-deterministic, and it is applied to non-directed graphs. Graph subisomorphism is an *NP*-complete problem, and no known polynomial time algorithm for finding a graph isomorphism exists. As a result, heuristic similarity measures have been implemented to compute the similarity between two graphs. Similarity can be computed by either cost-based measures or feature-based measures. In cost-based (tree edit) approaches (e.g. [23], [24]), some functions are used to determine the minimum cost to transform one graph to the other. In feature-based approaches (e.g. [25]), the similarity is calculated by extracting a set of structural features and comparing them.

In general, we can represent features in CAD models as directed and labeled graphs. Our effort here is on how to map the features between two different systems to allow interoperable data exchange. Since the definitions of features vary in different CAD systems, it is highly likely that the same feature is represented by different graph structures. To map a feature from one graph structure to another, we need to take a subgraph isomorphism approach to determine the degree of similarity between the two graphs depending on both the structure and the label. The preprocessing of type checking can accelerate the subgraph matching.

3 HYBRID SEMANTIC FEATURE MODELING

The hybrid semantic feature model we proposed in [1] is used to represent implicit and explicit features in order to enable history-neutral feature construction. It is intended to capture feature semantics in a textual document and support feature mapping as we will discuss in Section 4. In the remainder of this section we summarize the hybrid semantic feature model, present a new formalization of that model and describe its representation in RDF.

3.1 Hybrid Semantic Feature Definition

A hybrid semantic feature is represented by a directed, labeled, and attributed graph. The center of a feature graph is the name of feature, surrounded by several attributes which represent geometric and topological information to define the feature. Our model is associated with eight types of attribute, and we classify them as *individual, interfacial,* or *alias*. *Individual attributes* are used to characterize the attributes that only belong to one feature, including Parameter, Sketch and BooleanSign. On the other hand, *interfacial attributes* are supplied to define the boundaries of features that belong to more than one feature. Four interfacial attributes are Point, Line, Surface, and SolidBody. The *alias* attribute is used to capture the possible alias name of a feature, either from different systems or from different naming methods. For example, a long object of a fixed cross-sectional profile is referred to as *Extruded Boss/Base* in SolidWorks® software and Extrude in its API. But the same feature is referred as Protrusion in SolidEdge®. Thus *Extruded Boss/Base* and *Protrusion* are the aliases of the same feature in different systems. This alias attribute in the model adds more information and facilitates the mapping process, which will be discussed in Section 4.

Figure 2 is a simple illustration of how a *Extrude* feature is represented in a feature graph. The darker blue rectangles are the *interfacial* attributes whereas the lighter blue rectangles are the individual attributes. The center node of the graph is the feature node, with the name "Feature_Extrude_1". The direction

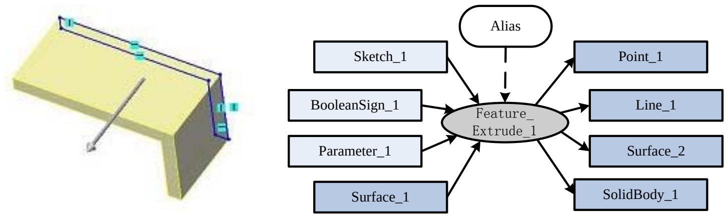


Figure 2. A feature graph example of feature *Extrude* from SolidWorks®.

of an edge denotes *reference* or *inclusion* dependency between two nodes, which also differentiates the *explicit* attributes from *implicit* attributes of the center feature. *Explicit* attributes are used to define a feature, whereas *implicit* attributes are evaluated from this feature and serve as references for other dependent features. For example, "Feature_Extrude_1" in Figure 2 was created by the extrusion of a sketch on a surface, so its explicit attributes are "Surface_1" (datum plane), "Sketch_1" (profile sketch), "Parameter_1" (extrude depth) and "BooleanSign_1" (Union). At the same time, the possible implicit attributes of "Feature_Extrude_1" such as "Point_1", "SolidBody_1", "Surface_2", and "Line_2" are used to form other features. They build direct connections between "Feature_Extrude_1" and other features.

Based on the interrelationship between features, the hybrid semantic model is the assembly of all its features. It provides a comprehensive description of the model with a multi-resolution graph. In the hybrid semantic model, we define three different levels of information. A Level One graph captures the overall feature dependency which only shows the relationships of features. Closely related features can be further grouped and form composite features at Level One. Level Two extends the dependency relationships with interfacial attributes which represent the topology information. And Level Three represents the complete definition of features by adding individual attributes.

High-level feature graphs provide more abstract information whereas detailed information is fully described in low-level models. The multi-resultion model also helps in the feature mapping process. Theoretically, the mapping can be performed at any level of feature graphs. High-level graphs with reduced complexity can improve the efficiency of mapping. For the feature creation purpose, we usually need Level-Three graphs with complete information of definitions.

3.2 Formal Model of Hybrid Semantic Features

Although section 3.1 informally describes the hybrid semantic feature model, in order to connect the formal type theory with the CAD interoperability algorithms presented in section 4, *formal* definitions of such models are needed. This section formally defines such models, and then defines feature substitution in a model. These formal definitions lead to safety theorems related to our conversion process.

Recall from section 3.1 that a CAD model M is graph of features and attributes. All details of a model are captured at level 3 resolution, so our definition is based on this resolution.

Definition 1. Features are represented using the format: `<FName>[<aname1>:<type1>, ...,<anameN>:<typeN>]`; for example, `ExtrudedHole[radius : RealSolidEdge; depth : IntSolidEdge]` denotes a feature named `ExtrudedHole` that has two attributes named `radius` and `depth`, that are of type `RealSolidEdge` and `IntSolidEdge`, respectively.

Definition 2. Let *Features* be the set of all features and *Attributes* be the set of all attributes. The set of attribute of a feature is defined by the function $\Gamma : \mathit{Features} \rightarrow 2^{\mathit{Attributes}}$ such that $\Gamma(F[x_1 : t_1, x_2 : t_2, \ldots, x_n : t_n]) = \{(x_1,t_1),(x_2,t_2),\ldots,(x_n,t_n)\}$.

Definition 3. A model $M = (V,E,F,A)$ is a graph such that V is the set of vertices in M, E is the set of edges in M, and:

$$
\begin{aligned}
F &= \text{the set of all features in model } M \\
A &= \bigcup_{f \in F} \Gamma(f) \quad \text{(attributes of features in } M\text{)} \\
V &= F \cup A \\
E &\subseteq V \times V
\end{aligned}
$$

For clarity, when dealing with multiple models, we use function-application notation to denote a set with respect to a model; for example, the features and vertices of model M can be denoted $F(M)$ and $V(M)$, respectively.

Definition 4. A *simple* path in a graph is a path that does not visit any vertex twice. The notation $(v_1 \rightarrow \ldots \rightarrow v_2) \in M$ denotes there exists a simple path from v_1 to v_2 in M; $(v_1 \rightarrow \ldots v_2 \ldots \rightarrow v_3) \in M$ denotes that there exists a simple path from v_1 to v_3 in M such that v_2 is an intermediate vertex on that path.

Definition 5. The set of interfacial attributes of M is $I(M) = \{a \in A(M) \mid \exists f_1, f_2 \in F(M) \text{ s.t. } (f_1 \rightarrow \ldots a \ldots \rightarrow f_2) \in M\}$.

Definition 6. The level 2 graph of M is $L2(M)$, where:

$$
\begin{aligned}
L2(M) &= (V_2,E_2,F_2,A_2) \\
F_2 &= F(M) \\
A_2 &= I(M) \\
V_2 &= F_2 \cup A_2 \qquad = F(M) \cup I(M) \\
E_2 &= E(M) \cap (V_2 \times V_2)
\end{aligned}
$$

Definition 7. The level 1 graph of M is $L1(M)$, where:

$$\begin{aligned} L1(M) &= (V_1, E_1, F_1, A_1) \\ F_1 &= F(M) \\ A_1 &= \emptyset \\ V_1 &= F(M) \\ E_1 &= \{(f_1, f_2) \in F \times F \mid (f_1 \to \ldots \to f_2) \in L2(M)\}. \end{aligned}$$

Definition 8. The substitution of a feature f for another feature f' in a model $M = (V, E, F, A)$ under the attribute conversion function $g : \Gamma(f) \to \Gamma(f')$ denoted $g \vdash \{f'/f\}M$ is the model $M' = (V', E', F', A')$ s.t.:

$$\begin{aligned} F' &= (F - \{f\}) \cup \{f'\} \\ A' &= (A - \Gamma(f)) \cup \Gamma(f') \\ V' &= F' \cup A' \\ E' &= \{(h_f(v), h_f(v')) \mid (v, v') \in E\}, \text{ where:} \end{aligned}$$

$h_f : V(M) \to V(M')$ such that

$$h_f(v) = \begin{cases} f' & \text{if } v = f \\ g(v) & \text{if } v \in \Gamma(f) \\ v & \text{otherwise} \end{cases}$$

In words, substitution of f' for f in M is basically a graph transformation, where f is removed, f' is added, and each edge (v, v') is updated to $(h_f(v), h_f(v'))$ to be between the appropriate vertices.

3.2.1 Model Replacement

This section defines when a model can be replaced with another. This formal definition enables one to determine whether a feature substitution performed in a model results in a model that is *equivalent* to or more *specialized* than the original.

Recall that two graphs G and G' are isomorphic when $\exists h : V(G) \xrightarrow[onto]{1-1} V(G')$ s.t $(v, v') \in E(G) \iff (h(v), h(v')) \in E(G')$; if h was only one-to-one, then G is isomorphic to a subgraph of G'.

Definition 9. Two models M and M' are equivalent, denoted $M \equiv M'$, iff $\exists h : V(M) \xrightarrow[onto]{1-1} V(M')$ s.t $(v, v') \in E(M) \iff (h(v), h(v')) \in E(M')$, where $v \in A(M) \implies v \equiv h(v)$.

Definition 10. If h from definition 9 was only one-to-one, then M is *subequivalent* to M', denoted $M \subseteq_{\equiv} M'$; in other words, M is equivalent to a submodel of M'.

Finally, M' can be replace M, if M' is a specialization of M, which is denoted $M' <: M$.

Definition 11. $M' <: M \iff L2(M) \equiv L2(M')$ and $M \subseteq_{\equiv} M'$. The condition $L2(M) \equiv L2(M')$ ensures that relationships between features in the design of the model are not violated by adding interfacial attributes. Figure 3 gives an example of two models M_1 and M_2, where $M_1 \subseteq_{\equiv} M_2$ but their level 2 models are not equivalent.

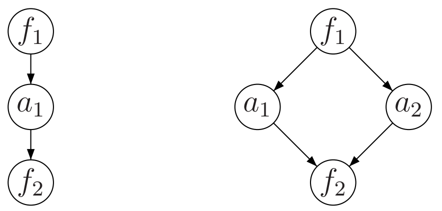


Figure 3. f_1 and f_2 are features. a_1 and a_2 are attributes. Left model is subequivalent to right model but their level two graphs are not equivalent.

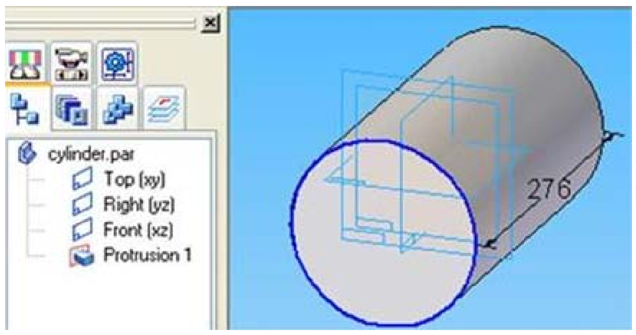
3.3 Hybrid Semantic Feature Representation

To model and represent the hybrid semantic features in a way that facilitates documentation and mapping, we use the resource description framework (RDF) [26] to represent features. RDF format was built on top of the extensible markup language (XML) syntax. XML enables syntax-level interoperability. RDF represents relationships in a semantic model as subject-predicate-object triples. More importantly, RDF documents are useful to describe the graph-structure information, which simplifies the mapping process.

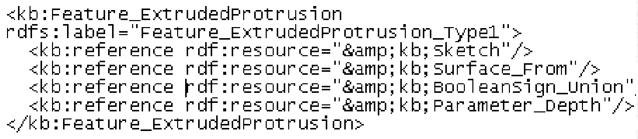
Two levels of feature information, class level and instance level, can be captured in RDF documents. As shown in the example of Figure 4, the class level information defines an extrusion feature. It only contains the feature name and the types of attributes that define the feature (Figure 4-b). At the instance level, the RDF extends the class level RDF by adding attribute nodes as well as their parameters, which can be used to create a concrete feature in CAD systems (Figure 4-c). A complete CAD model is either a composite RDF file with instance-level features or a collection of instance level RDF files. Based on the class-level RDF feature documents, we can also create feature libraries corresponding to CAD systems. The semantic information of features is recorded in the library documents. The class-level RDF files are useful in static mappings and the instance-level RDF files are used in dynamic mappings, which will be introduced in Section 4.1.

4 SEMANTIC FEATURE MAPPING

A feature mapping algorithm is developed. The algorithm compares two feature graphs by both labels and structures and calculates the semantic similarity. Two similarity measures are considered through two stages: *label matching* and *attribute matching*. Label matching simply compares the names of two features as well as their aliases and returns if they are equal. In the attribute matching, the number of explicit attributes of the

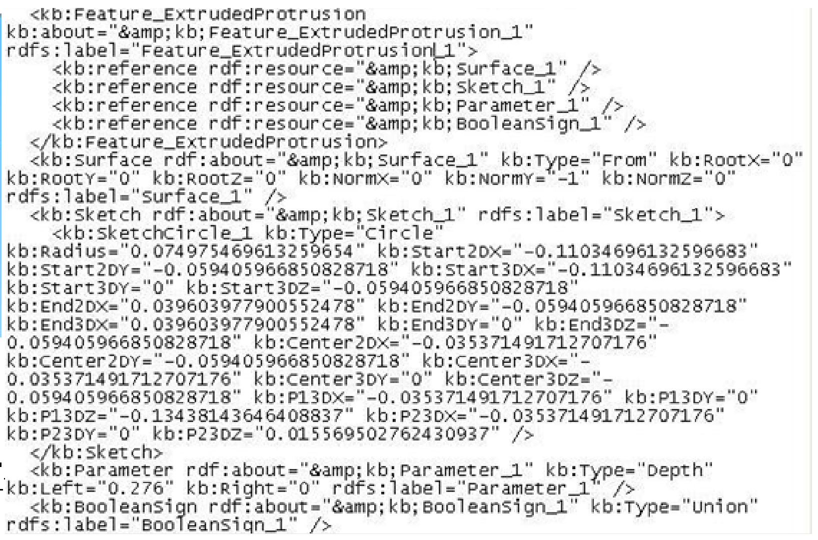


a: An ExtrudedProtrusion Feature in SolidEdge



```
<kb:Feature_ExtrudedProtrusion
rdfs:label="Feature_ExtrudedProtrusion_Type1">
  <kb:reference rdf:resource="&amp;kb;Sketch"/>
  <kb:reference rdf:resource="&amp;kb;Surface_From"/>
  <kb:reference rdf:resource="&amp;kb;BooleanSign_Union"
  <kb:reference rdf:resource="&amp;kb;Parameter_Depth"/>
</kb:Feature_ExtrudedProtrusion>
```

b: Class Level of ExtrudedProtrusion Feature



```
  <kb:Feature_ExtrudedProtrusion
kb:about="&amp;kb;Feature_ExtrudedProtrusion_1"
rdfs:label="Feature_ExtrudedProtrusion_1">
    <kb:reference rdf:resource="&amp;kb;Surface_1" />
    <kb:reference rdf:resource="&amp;kb;Sketch_1" />
    <kb:reference rdf:resource="&amp;kb;Parameter_1" />
    <kb:reference rdf:resource="&amp;kb;BooleanSign_1" />
  </kb:Feature_ExtrudedProtrusion>
  <kb:Surface rdf:about="&amp;kb;Surface_1" kb:Type="From" kb:RootX="0"
kb:RootY="0" kb:RootZ="0" kb:NormX="0" kb:NormY="-1" kb:NormZ="0"
rdfs:label="Surface_1" />
  <kb:Sketch rdf:about="&amp;kb;Sketch_1" rdfs:label="Sketch_1">
    <kb:SketchCircle_1 kb:Type="Circle"
kb:Radius="0.074975469613259654" kb:Start2DX="-0.11034696132596683"
kb:Start2DY="-0.059405966850828718" kb:Start3DX="-0.11034696132596683"
kb:Start3DY="0" kb:Start3DZ="-0.059405966850828718"
kb:End2DX="0.039603977900552478" kb:End2DY="-0.059405966850828718"
kb:End3DX="0.039603977900552478" kb:End3DY="0" kb:End3DZ="-
0.059405966850828718" kb:Center2DX="-0.035371491712707176"
kb:Center2DY="-0.059405966850828718" kb:Center3DX="-
0.035371491712707176" kb:Center3DY="0" kb:Center3DZ="-
0.059405966850828718" kb:P13DX="-0.035371491712707176" kb:P13DY="0"
kb:P13DZ="-0.13438143646408837" kb:P23DX="-0.035371491712707176"
kb:P23DY="0" kb:P23DZ="0.015569502762430937" />
  </kb:Sketch>
  <kb:Parameter rdf:about="&amp;kb;Parameter_1" kb:Type="Depth"
kb:Left="0.276" kb:Right="0" rdfs:label="Parameter_1" />
  <kb:BooleanSign rdf:about="&amp;kb;BooleanSign_1" kb:Type="Union"
rdfs:label="BooleanSign_1" />
```

c: Instance Level of ExtrudedProtrusion Feature

Figure 4. A RDF example for a feature graph.

feature is counted and the types of the attributes are then listed. Attribute similarity is then measured according to the number of the same type of attributes that are exactly matched.

The semantic mapping procedure for a complete CAD model can be summarized as algorithm 1. The algorithm starts with label matching which checks if the labels are equal. The source feature is mapped to the target feature with the equal label directly. In the case that a label matching is found, the attribute matching is applied to determine the degree of similarity. If no feature with the same label is found, then the attribute matching is applied between the source feature and all of the target features. The source feature is then mapped to the target feature that holds the highest degree of similarity. The label matching step reduces the complexity of attribute matching step if aliases of features are found directly.

Two kinds of degree of similarity are calculated, which are (*Source_sim*) and (*Target_sim*) respectively. The number of attributes in the source feature (*S_count*) and that in the target feature (*T_count*) are counted first. A comparison between the types of attributes is performed, and a counter (*sim_count*) is incremented with each matching attribute. Then similarity is measured as

$$Source\_sim = \frac{sim\_count}{S\_count} \qquad Target\_sim = \frac{sim\_count}{T\_count}$$

The overall degree of similarity between the source feature and target feature is the minimum value between *Source_sim* and *Target_sim*.

After a source feature is mapped to a target feature, the name of target feature will be added as an alias to the source feature to eliminate possible redundant attribute matching. If another instance of the same source feature is found in the next mapping, label matching will find a target feature with the alias. This reduces the matching complexity.

Our mapping algorithm is tailored for the feature structures as labeled graphs. The label matching step as the first filtering process can reduce computational cost when mapping features. Furthermore, the type of attributes can only be one of the eight described in Section 3.1. This also reduces the computational complexity and gives our algorithm an advantage over the general graph matching algorithms summarized in Section 2.3.

Algorithm 1 The feature mapping algorithm for a CAD model

```
Input: An RDF document to represent a model as a graph structure, where each node corresponds to a feature
1: Starting with the first node as the Source Feature
2: Search for a feature in the set of target features to match the label of the Source Feature
3: if Found then
4:     Compare the attributes of the two features
5:     Calculate Source_sim and Target_sim
6:     Mapping the source feature to the target feature
7: else
8:     Search for a feature in the set of target features with the same number and type of attributes
9:     Calculate Source_sim and Target_sim determine the overall degree of similarity
10:    Map the source feature to the target feature with the highest overall degree of similarity
11: end if
12: Traverse through the RDF document and repeat steps 2-11 for each node
```

There are two mapping scenarios, *static mapping* and *dynamic mapping*. As illustrated in Figure 5-a, in static mappings, class-level RDF files in libraries are used. If the class-level feature definitions corresponding to two systems, "System 1" and "System 2", are matched through the mapping process, a translator may be produced for one-to-one mappings, which is similar to the one used in current industry practice. This translator uses an extendable database which contains the labels and attributes of all features from both systems. If the static mapping is extended for three or more systems, the RDF library files of the third system can be added to the translator database so that the translator is updated. Once the database of the translator is created, it can be used repeatedly to generate RDF feature documents for existing CAD systems unless an update is required.

The main advantage of the proposed approach is dynamic mapping. It is the mapping between an instance-level RDF document from one system and the class-level RDF library in another system, as illustrated in Figure 5-b. An instance-level RDF document for the second system will be created. Different from static mapping, dynamic mapping does not require the library of the first system in this 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 RDF document and uploaded to the server. The dynamic mapping process can locate the specific feature in the original complete instance-level RDF document and update the parameters.

5 TYPE-CHECKING CAD FEATURE MAPPINGS

Type-checking feature mappings leads to an automated system that can verify if certain properties are preserved when converting a CAD model from one system to another. When converting a model to an equivalent model in another system, a natural question that arises is how to determine if one feature can be used in place of another. One such scenario occurs when one feature is a *specialization* of another; in the following section, we formally define this relationship.

5.1 Equivalent subset relation

The relationship between attribute sets of distinct features can signal if one feature can be used in place of another. Using the conventional subset relation $\subseteq$ between attribute sets has the restriction of requiring an exact match between attributes from two sets. However, equivalence relations are enough for valid conversions in many interoperability scenarios, so being dependent on exact matches may be too restrictive. Therefore, we will use a more general subset relation, denoted $\subseteq_\equiv$ and called the *equivalent-subset relation*, that is based on the conventional definition $\subseteq$. $A \subseteq_\equiv B$ can be read as "A is equivalent to a subset of B" or "A is an equivalent subset of B" The $\subseteq_\equiv$ relation will be used to reason about the sets of attributes referred to by two features.

The motivation behind the equivalent-subset relation is as follows. If `(radius, RealSolidEdge)` $\equiv$ `(radius, RealSolidWorks)`, then one could conclude that the feature `ExtrudedHole[radius : RealSolidEdge, depth : IntSolidEdge]` is a specialization of the feature `Hole[radius : RealSolidWorks]`. Hence, any place where a model uses an instance of the feature `Hole[radius : RealSolidWorks]` should be able to use an instance of `ExtrudedHole[radius : RealSolidEdge, depth : IntSolidEdge]` by converting its `radius` attribute into the appropriate representation. In general, a feature with attribute set B is a specialization of another feature with attribute set A if $\exists g : A \xrightarrow{1-1} B, \forall x \in A, x \equiv g(x)$. In words, either A is empty or there must be a mapping from elements in A to equivalent elements in B such that no two elements from A are mapped to the same element in B. Although the condition of the existence of a function between A and B is rigorous, inductive rules of the form $A \subseteq_\equiv B$ are given in the remainder of the section that capture this condition because they lead to proofs that can be derived mechanically by an automated theorem prover.

First, the equivalence relation $\equiv$ has the conventional reflexive, symmetric, and transitive properties. Figure 6 presents the inductive definition of the equivalent subset relation. Rule SUB_EMPTY recognizes that every set contains the empty set, which, of course, is equivalent to the empty set. Adding elements to an "equivalent super set" does not contradict the $\subseteq_\equiv$ relation, which corresponds to rule SUB_SUPER. SUB_ADD allows one to add elements to both sets in a way that preserves the relation. Rules SUB_REFL and SUB_TRAN state that the $\subseteq_\equiv$ relation preserves the reflexive and transitive properties of $\subseteq$. Figure 7 illustrates an application of the rules with an example $\subseteq_\equiv$ judgment and a derivation of the judgment.

The following lemma shows that given $A \subseteq_\equiv B$, there is a mapping that enables replacing attributes in A with equivalent attributes in B.

Lemma 1. *Let A and B be sets. If $A \subseteq_\equiv B$, then $\exists g : A \xrightarrow{1-1} B, \forall x \in A, x \equiv g(x)$.*

Proof. First, functions are represented as a set of ordered pairs $g = \{(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)\}$ such that all x_i are pairwise distinct and $g(x_i) = y_i$. The domain of g is $dom(g) = \{x_1,x_2,\dots,x_n\}$. If $S \subseteq dom(g)$, then the image of g on S is $g(S) = \{g(s) \mid s \in S\}$. The range of g is $range(g) = g(dom(g))$.

Case SUB_EMPTY: The function $g = \emptyset$ is a one-to-one function from $\emptyset$ to B, since it does not map two elements to the same element in B. □

Case SUB_SUPER: Suppose the inductive hypothesis that $A \subseteq_\equiv B$ and that $\exists g : A \xrightarrow{1-1} B, \forall x \in A, x \equiv g(x)$. Adding the

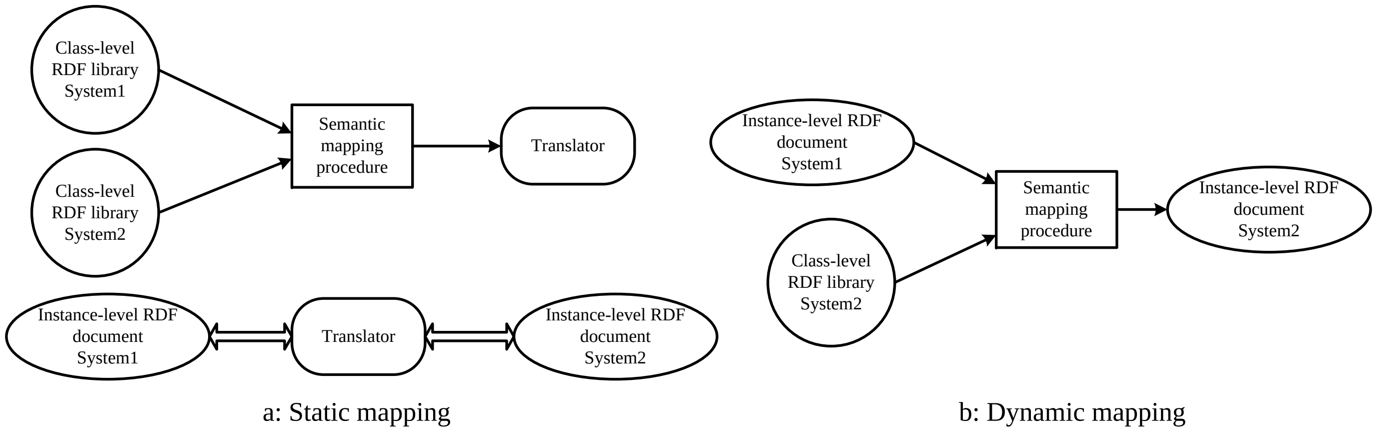

Class-level RDF library System1
Class-level RDF library System2
Semantic mapping procedure
Translator
Instance-level RDF document System1
Translator
Instance-level RDF document System2
a: Static mapping
Instance-level RDF document System1
Class-level RDF library System2
Semantic mapping procedure
Instance-level RDF document System2
b: Dynamic mapping

Figure 5. The Diagrams of Static Mapping and Dynamic Mapping.

$$\frac{}{\emptyset \subseteq_\equiv A}\ \texttt{SUB\_EMPTY} \qquad \frac{A \subseteq_\equiv B}{A \subseteq_\equiv B \cup C}\ \texttt{SUB\_SUPER} \qquad \frac{A \subseteq_\equiv B \quad x \equiv x' \quad x \notin A \quad x' \notin B}{A \cup \{x\} \subseteq_\equiv B \cup \{x'\}}\ \texttt{SUB\_ADD}$$

$$\frac{}{A \subseteq_\equiv A}\ \texttt{SUB\_REFL} \qquad \frac{A \subseteq_\equiv B \quad B \subseteq_\equiv C}{A \subseteq_\equiv C}\ \texttt{SUB\_TRAN}$$

Figure 6. Equivalent Subset Relation

Derivation A:

$$\dfrac{\dfrac{}{\emptyset \subseteq_\equiv \{\texttt{Real}_{Edge}\}}\ \texttt{SUB\_EMPTY} \quad \texttt{Str}_{Works} \equiv \texttt{Str}_{Edge} \quad \texttt{Str}_{Works} \notin \emptyset \quad \texttt{Str}_{Edge} \notin \{\texttt{Real}_{Edge}\}}{\underbrace{\{\texttt{Str}_{Works}\}}_{\emptyset \cup \{\texttt{Str}_{Works}\}} \subseteq_\equiv \underbrace{\{\texttt{Str}_{Edge}, \texttt{Real}_{Edge}\}}_{\{\texttt{Real}_{Edge}\} \cup \{\texttt{Str}_{Edge}\}}}\ \texttt{SUB\_ADD}$$

$$\dfrac{\dfrac{\text{By derivation } A \text{ above}}{\{\texttt{Str}_{Works}\} \subseteq_\equiv \{\texttt{Str}_{Edge}, \texttt{Real}_{Edge}\}} \quad \texttt{Int}_{Works} \equiv \texttt{Int}_{Edge} \quad \texttt{Int}_{Works} \notin \{\texttt{Str}_{Works}\} \quad \texttt{Int}_{Edge} \notin \{\texttt{Str}_{Edge}, \texttt{Real}_{Edge}\}}{\underbrace{\{\texttt{Int}_{Works}, \texttt{Str}_{Works}\}}_{\{\texttt{Str}_{Works}\} \cup \{\texttt{Int}_{Works}\}} \subseteq_\equiv \underbrace{\{\texttt{Int}_{Edge}, \texttt{Str}_{Edge}, \texttt{Real}_{Edge}\}}_{\{\texttt{Str}_{Edge}, \texttt{Real}_{Edge}\} \cup \{\texttt{Int}_{Edge}\}}}\ \texttt{SUB\_ADD}$$

Figure 7. Example equivalent subset judgment and its derivation

elements from C to the range of g does not change that g is one-to-one, so it is also the case that $g : A \xrightarrow{1-1} B \cup C, \forall x \in A, x \equiv g(x)$. $\square$

Case SUB_ADD: Suppose the inductive hypothesis that $A \subseteq_\equiv B$, $x \notin A$, $x' \notin B$, $x \equiv x'$, and that $\exists g : A \xrightarrow{1-1} B, \forall z \in A, z \equiv g(z)$. Since $x \notin A$, $g \cup \{(x,x')\}$ is a set that is still a function. $g \cup \{(x,x')\}$ remains an injective function because $x' \notin B = \mathit{range}(g) \implies g$ does not map any element other than x to x'. Lastly, since $x \equiv x'$, $g \cup \{(x,x')\}$ only maps elements to equivalent elements. $\therefore g \cup \{(x,x')\} : (A \cup \{x\}) \xrightarrow{1-1} (B \cup \{x'\}), \forall z \in (A \cup \{x\}), z \equiv g(z)$ $\square$

Case SUB_REFL: The identity function $I : A \to A$, $I(x) = x, \forall x \in A$ is a one-to-one function and $x \equiv x = I(x)$, since $\equiv$ is reflexive. $\square$

Case SUB_TRAN: Suppose $A \subseteq_\equiv B$, $B \subseteq_\equiv C$, $\exists g : A \xrightarrow{1-1} B$ s.t. $\forall x \in A, x \equiv g(x)$, and $\exists h : B \xrightarrow{1-1} C$ s.t. $\forall x \in B, x \equiv h(x)$. Define $h \circ g : A \to C$ s.t. $h \circ g(x) = h(g(x)), \forall x \in A$. Since h and g are one-to-one, then the composition of h and g, $h \circ g$ is one-to-one. Also, $\forall x \in A, x \equiv g(x) \equiv h(g(x))$. $\therefore h \circ g : A \xrightarrow{1-1} C$ s.t. $x \equiv h(g(x)) = h \circ g(x), \forall x \in A$. $\square$

Definition 12. The equivalent subset relation leads to a formal definition of when one feature is a specialization of another. The notation $f' <: f$ denotes that feature f' is a specialization of feature f. The following rule defines the specialization judgment.

$$\frac{\Gamma(f) \subseteq_{\equiv} \Gamma(f')}{f' <: f} \texttt{ SPECIAL}$$

5.2 Safety Theorems Regarding CAD Model Conversions

The definitions presented enable reasoning about CAD model conversions, where models are converted by substituting features for others. The first theorem reasons about a feature substitution for *any* CAD model. The second theorem reasons about a feature substitution in a *given* CAD model. Since a particular model M may not "use" all of a feature's attributes, we define the notation $M(f) = \Gamma(f) \cap A(M)$, which are the attributes of f that are used in M.

Theorem 1 (Static mapping). *If $f' <: f$, then $\exists g : \Gamma(f) \to \Gamma(f')$ s.t. $g \vdash \{f'/f\}M <: M$, for any model M.*

Theorem 2 (Dynamic mapping). *For a given model M, if $M(f) \subseteq_{\equiv} \Gamma(f')$, then $\exists g : M(f) \to \Gamma(f')$ s.t. $g \vdash \{f'/f\}M <: M$.*

Proofs of the theorems are given below. If neither of the two premises hold for theorems 1 and 2, then using a conversion function $g : M(f) \to \Gamma(f')$ will require a dynamic check or human intervention to ensure that $g \vdash \{f'/f\}M <: M$.

Proof of Theorem 1. Suppose $f' <: f$. The only way this assumption holds is if there exists a derivation of $\Gamma(f) \subseteq_{\equiv} \Gamma(f')$. By lemma 1, $\exists g : \Gamma(f) \xrightarrow{1-1} \Gamma(f'), \forall x \in \Gamma(f), x \equiv g(x)$. Let $M' = g \vdash \{f'/f\}M$.

Consider $h_f : V(M) \to V(M')$ from definition 8 and the partition of its domain: $\{f\}, \Gamma(f), V(M) - (\{f\} \cup \Gamma(f))$. The images of each partition:

$$\begin{aligned} h_f(\{f\}) &= \{f'\} \\ h_f(\Gamma(f)) &= g(\Gamma(f)) \subseteq \Gamma(f') \\ h_f(V(M) - (\{f\} \cup \Gamma(f))) &\subseteq V(M) \end{aligned}$$

clearly share no elements and are mutually disjoint. Hence, if h_f is one-to-one on each partition, then h_f is one-to-one overall.

h_f is trivially one-to-one on {f}. h_f on $(V(M) - (\{f\} \cup \Gamma(f))$ is equal to the identity function and thus, is one-to-one on this partition. On $\Gamma(f)$, $h_f = g$; since g is one-to-one on $\Gamma(f)$, so is h_f.

$\therefore h_f : V(M) \xrightarrow{1-1} V(M')$.

By definition 8, $(v, v') \in E(M) \iff (h(v), h(v')) \in E(M')$, so M is subisomorphic to M'. Also, for each $a \in A(M)$ either $h(a) = a$ if $a \in (A(M) - \Gamma(f))$ or $h(a) = g(a) \equiv a$ if $a \in \Gamma(f)$.

$\therefore \forall a \in A(M), a \equiv h(a)$ and $M \subseteq_{\equiv} M'$.

Now suppose $a' \in I(M')$. Then $\exists f'_1, f'_2 \in F(M')$ s.t. $(f'_1 \to \ldots a' \ldots \to f'_2) \in M'$. So it must be the case that a' is incident to an edge in $E(M')$. By definition of $E(M')$, each end point of an edge in $E(M')$ is a vertex that was mapped to by h_f.

$\therefore a' \in range(h)$.

Hence, h_f maps to all interfacial attributes and features in M'. Since the only attributes in level 2 graphs are interfacial, then it must be the case that $h_f : V(L2(M)) \underset{onto}{\xrightarrow{1-1}} V(L2(M'))$. Lastly, h_f is isomorphic between the vertices of its domain and range, so it follows that $L2(M) \equiv L2(M')$. □

Proof of Theorem 2. Proof of this theorem follows from theorem 1 by choosing a fixed model M. □

6 IMPLEMENTATION

We demonstrate the dynamic mapping process by the data sharing between SolidEdge® and SolidWorks®. An anchor joint is originally created in SolidEdge®. An instance-level RDF document of the anchor joint is created by extracting the names and attributes of features. The detailed information of attributes is also contained in this file, which is shown in Figure 8.

The dynamic mapping process is executed between the instance-level RDF document from source system and the class-level RDF library file of target system. Source features are compared with target features according to the semantic feature mapping algorithm presented in Section 4. Each source feature is mapped to the target feature with the same label or highest overall degree of similarity, as shown in Figure 8. Depending on different Source_sim and Target_sim, there are four possible mapping results. Under each result, there is a translation process to translate the label and attributes from source feature to the target feature. A new instance-level RDF document is generated which can be used in target system directly.

Case 1: $Source\_sim = 1, Target\_sim = 1$, the source feature and target feature are equivalent. In this case, all attributes of source feature are same as those of the target feature which means that the definition of source feature and target feature is identical. Hence, all the attributes of such features can be used in target system directly. The only thing that needs to be translated is the label, when the label is unmatched. Five features in Figure 8 are in this case, such as Feature_ExtrudedProtrusion_1, both Source_sim and Target_sim are equal to 1, and the label is unmatched, which means the attributes in source feature are the same as in the target feature, Feature_Extrude_Type3. In the translation, the feature label is changed to Feature_Extrude_1 and all the attributes are kept the same in the new RDF document.

Case 2: $Source\_sim < 1, Target\_sim < 1$, the source feature and target feature are not equivalent. This case is a general case that represents when the definition of source feature is different from the target feature. In this case, we need to focus on how to represent all the attributes in target feature using attributes in source feature. Three kinds of attributes are defined here, the attributes contained in both source feature and target feature is named as *common attributes*; the attributes only contained in source feature are *redundant attributes*, and the attributes only contained in target feature are *missing attributes*.

For the common attributes, they can be used in both source feature and target feature directly. For the redundant attributes, they are not used in the target system, but they will be very useful for calculating the missing attributes. For the missing attributes, they have to be calculated by several attributes in the source feature. The calculation algorithm is based on some fundamental principles among attributes, such as three non-collinear points can determine a plane. These principles will be incorporated in the mapping algorithm. There are two examples of this case in Figure 8, Feature_ExtrudedProtrusion_2 and Feature_ExtrudedCutout_3. The previous feature is created by extruding a sketch from the sketch plane to the next plane in the model. Hence its attributes are Sketch, Surface_From, and three BooleanSign representing ToNext, Union, and OneDirection. In the instance-level RDF document, BooleanSign does not contain geometry information, hence there is not enough geometry information in the Feature_ExtrudedProtrusion_2 to recreate the feature, which will probably cause some errors in the translation. We add an artificial reference node to this feature, $a\_reference$ to provide enough geometry information of the model, in this example, the a_reference is with the type Surface_To which is calculated by Surface_From and Boolean_ToNext. By using Sketch, Surface_From and Surface_To, this ExtrudedProtrusion can be fully defined. Then, in the mapping process, this feature is mapped to Feature_Extrude_Type2, and the artifical reference can be used directly in the target feature. Similar to Feature_ExtrudedProtrusion_2, Feature_ExtrudedCutout_3 is extruded from a plane created by three points. There is enough geometry information contained in the source feature hence no artificial attribute is introduced. The missing attribute in the target feature, Surface_From is calculated from those three points.

Case 3: $Source\_sim = 1, Target\_sim < 1$, the source feature is an equivalent subset of the target feature. In this case, the number of common attributes is equal to the number of attributes in source feature but less than that in target feature which means there are some missing attributes to fully define the target feature. These missing attributes need to be retrieved from the attributes in source feature by the method presented in Case 2.

Case 4: $Source\_sim < 1, Target\_sim = 1$, the target feature is an equivalent subset of the source feature. This case is opposite to Case 3, there are some redundant attributes in source feature. All the common attributes are kept, but the redundant attributes in source feature are removed.

By the translation, a new RDF document is created so that the anchor joint can be built or further modified in SolidWorks®, which realizes the collaboration between SolidEdge® and SolidWorks®.

7 SUMMARY AND FUTURE WORK

Interoperability is a major challenge for distributed collaborative engineering. In this paper we have presented results demonstrating the potential of static and dynamic feature mapping for enabling interoperability between CAD systems. We have described the hybrid semantic feature model, on which our approach is based, and presented a formalization of hybrid semantic features. We have used the formalization to give some rigorous definitions and to prove some important properties related to several aspects of feature mapping. In particular, our safety theorems establish the conditions under which feature mappings can be done completely automatically, and when some intervention by a design engineer may be required. Finally, we have reported on a successful prototype implementation of feature mapping that achieves interoperability between SolidEdge® and SolidWorks®.

Given these encouraging results, we plan to pursue further improvements in support for interoperability in distributed collaborative engineering environments. In the near term, we will produce an enhanced version of our automated feature mapping system and extend it to work with additional CAD systems such as ProEngineer®. In the longer term, we hope to extend our approach to interoperability to cover other phases of distributed collaborative engineering beyond CAD.

ACKNOWLEDGMENT

This work is supported by National Science Foundation Industry-University Cooperative Research Center for e-Design.

REFERENCES

[1] Hanayneh, L., Wang, Y., Wang, Y., Wileden, J., and Qureshi, K., 2008. "Feature Mapping Automation for CAD Data Exchange". *2008 ASME International Design Engineering Technical Conferences & The Computer and Information in Engineering Conference (IDETC/CIE2008)*(DETC2008-49671).

[2] Wang, Y., and Nnaji, B., 2006. "Document-Driven Design for Distributed CAD Services in Service-Oriented Architecture". *ASME Journal of Computing and Information Science in Engineering,* **6**(2), pp. 127–138.

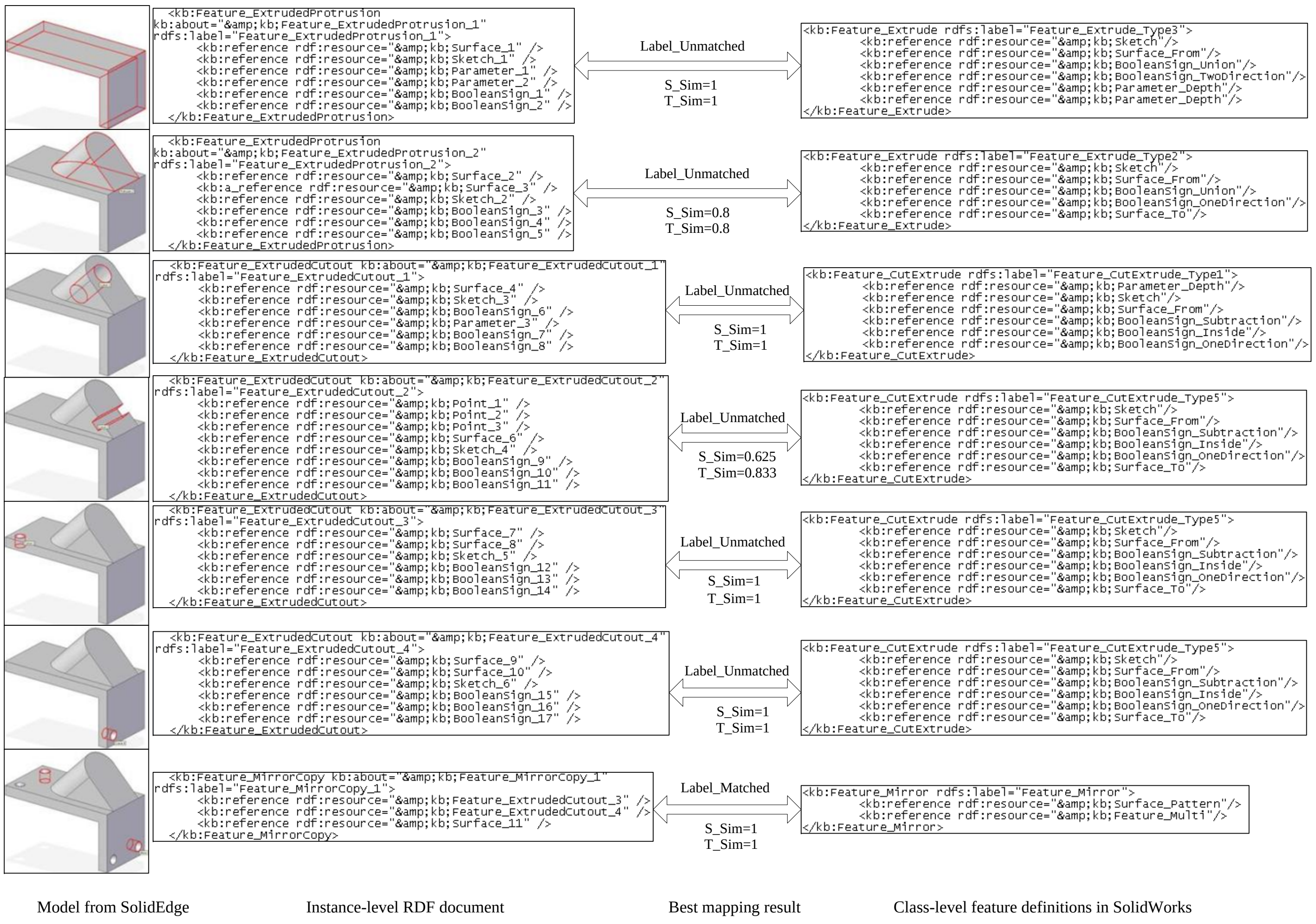
[3] Chen, X., and Hoffmann, C., 1995. "On Editability of Feature-based Design". *Comp.-Aided Des.,* **27**(12), pp. 905–914.

[4] Hoffmann, C., 1997. *CAD Systems Development*. Springer,Berlin, ch. EREP Project Overview, pp. 32–40.

[5] Shih, C., and Anderson, B., 1997. "A Design/Constraint Model to Capture Design Intent". *Proc. 4th ACM Symp. on Solid Modeling & Applications,* **27**(12), pp. 255–264.

[6] National Institute of Standards and Technology, 2003. Product Data Exchange Specification: The First Working Draft,NISTIR88-4004.

[7] Shah, J., and Mathew, A., 1991. "Experimental Inves-



```
  <kb:Feature_ExtrudedProtrusion
kb:about="&amp;kb;Feature_ExtrudedProtrusion_1"
rdfs:label="Feature_ExtrudedProtrusion_1">
      <kb:reference rdf:resource="&amp;kb;Surface_1" />
      <kb:reference rdf:resource="&amp;kb;Sketch_1" />
      <kb:reference rdf:resource="&amp;kb;Parameter_1" />
      <kb:reference rdf:resource="&amp;kb;Parameter_2" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_1" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_2" />
  </kb:Feature_ExtrudedProtrusion>
```

```
<kb:Feature_Extrude rdfs:label="Feature_Extrude_Type3">
        <kb:reference rdf:resource="&amp;kb;Sketch"/>
        <kb:reference rdf:resource="&amp;kb;Surface_From"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_Union"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_TwoDirection"/>
        <kb:reference rdf:resource="&amp;kb;Parameter_Depth"/>
        <kb:reference rdf:resource="&amp;kb;Parameter_Depth"/>
</kb:Feature_Extrude>
```

```
  <kb:Feature_ExtrudedProtrusion
kb:about="&amp;kb;Feature_ExtrudedProtrusion_2"
rdfs:label="Feature_ExtrudedProtrusion_2">
      <kb:reference rdf:resource="&amp;kb;Surface_2" />
      <kb:a_reference rdf:resource="&amp;kb;Surface_3" />
      <kb:reference rdf:resource="&amp;kb;Sketch_2" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_3" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_4" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_5" />
  </kb:Feature_ExtrudedProtrusion>
```

```
<kb:Feature_Extrude rdfs:label="Feature_Extrude_Type2">
        <kb:reference rdf:resource="&amp;kb;Sketch"/>
        <kb:reference rdf:resource="&amp;kb;Surface_From"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_Union"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_OneDirection"/>
        <kb:reference rdf:resource="&amp;kb;Surface_To"/>
</kb:Feature_Extrude>
```

```
  <kb:Feature_ExtrudedCutout kb:about="&amp;kb;Feature_ExtrudedCutout_1"
rdfs:label="Feature_ExtrudedCutout_1">
      <kb:reference rdf:resource="&amp;kb;Surface_4" />
      <kb:reference rdf:resource="&amp;kb;Sketch_3" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_6" />
      <kb:reference rdf:resource="&amp;kb;Parameter_3" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_7" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_8" />
  </kb:Feature_ExtrudedCutout>
```

```
<kb:Feature_CutExtrude rdfs:label="Feature_CutExtrude_Type1">
        <kb:reference rdf:resource="&amp;kb;Parameter_Depth"/>
        <kb:reference rdf:resource="&amp;kb;Sketch"/>
        <kb:reference rdf:resource="&amp;kb;Surface_From"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_Subtraction"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_Inside"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_OneDirection"/>
</kb:Feature_CutExtrude>
```

```
  <kb:Feature_ExtrudedCutout kb:about="&amp;kb;Feature_ExtrudedCutout_2"
rdfs:label="Feature_ExtrudedCutout_2">
      <kb:reference rdf:resource="&amp;kb;Point_1" />
      <kb:reference rdf:resource="&amp;kb;Point_2" />
      <kb:reference rdf:resource="&amp;kb;Point_3" />
      <kb:reference rdf:resource="&amp;kb;Surface_6" />
      <kb:reference rdf:resource="&amp;kb;Sketch_4" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_9" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_10" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_11" />
  </kb:Feature_ExtrudedCutout>
```

```
<kb:Feature_CutExtrude rdfs:label="Feature_CutExtrude_Type5">
        <kb:reference rdf:resource="&amp;kb;Sketch"/>
        <kb:reference rdf:resource="&amp;kb;Surface_From"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_Subtraction"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_Inside"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_OneDirection"/>
        <kb:reference rdf:resource="&amp;kb;Surface_To"/>
</kb:Feature_CutExtrude>
```

```
  <kb:Feature_ExtrudedCutout kb:about="&amp;kb;Feature_ExtrudedCutout_3"
rdfs:label="Feature_ExtrudedCutout_3">
      <kb:reference rdf:resource="&amp;kb;Surface_7" />
      <kb:reference rdf:resource="&amp;kb;Surface_8" />
      <kb:reference rdf:resource="&amp;kb;Sketch_5" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_12" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_13" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_14" />
  </kb:Feature_ExtrudedCutout>
```

```
<kb:Feature_CutExtrude rdfs:label="Feature_CutExtrude_Type5">
        <kb:reference rdf:resource="&amp;kb;Sketch"/>
        <kb:reference rdf:resource="&amp;kb;Surface_From"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_Subtraction"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_Inside"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_OneDirection"/>
        <kb:reference rdf:resource="&amp;kb;Surface_To"/>
</kb:Feature_CutExtrude>
```

```
  <kb:Feature_ExtrudedCutout kb:about="&amp;kb;Feature_ExtrudedCutout_4"
rdfs:label="Feature_ExtrudedCutout_4">
      <kb:reference rdf:resource="&amp;kb;Surface_9" />
      <kb:reference rdf:resource="&amp;kb;Surface_10" />
      <kb:reference rdf:resource="&amp;kb;Sketch_6" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_15" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_16" />
      <kb:reference rdf:resource="&amp;kb;BooleanSign_17" />
  </kb:Feature_ExtrudedCutout>
```

```
<kb:Feature_CutExtrude rdfs:label="Feature_CutExtrude_Type5">
        <kb:reference rdf:resource="&amp;kb;Sketch"/>
        <kb:reference rdf:resource="&amp;kb;Surface_From"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_Subtraction"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_Inside"/>
        <kb:reference rdf:resource="&amp;kb;BooleanSign_OneDirection"/>
        <kb:reference rdf:resource="&amp;kb;Surface_To"/>
</kb:Feature_CutExtrude>
```

```
  <kb:Feature_MirrorCopy kb:about="&amp;kb;Feature_MirrorCopy_1"
rdfs:label="Feature_MirrorCopy_1">
      <kb:reference rdf:resource="&amp;kb;Feature_ExtrudedCutout_3" />
      <kb:reference rdf:resource="&amp;kb;Feature_ExtrudedCutout_4" />
      <kb:reference rdf:resource="&amp;kb;Surface_11" />
  </kb:Feature_MirrorCopy>
```

```
<kb:Feature_Mirror rdfs:label="Feature_Mirror">
        <kb:reference rdf:resource="&amp;kb;Surface_Pattern"/>
        <kb:reference rdf:resource="&amp;kb;Feature_Multi"/>
</kb:Feature_Mirror>
```

Figure 8. Dynamic mapping of part Anchor Joint between SolidEdge$^{®}$ and SolidWorks$^{®}$.

tigation of The STEP Form-Feature Information Model". *Comp.-Aided Des.,* **23**(4), pp. 282–296.

[8] Kim, J., Pratt, M. J., Iyer, R. G., and Sriram, R. D., 2008. "Standardized data exchange of CAD models with design intent". *Comp.-Aided Des.,* **40**(7), pp. 760–777.

[9] Rappoport, A., Spitz, S., and Etzion, M., 2006. *Lecture Notes in Computer Science*. Springer,Berlin, ch. Two-Dimensional Selections for Feature-Based Data Exchange, pp. 325–342.

[10] Choi, G., Mun, M., and Han, S., 2002. "Exchange of CAD Part Models Based on the Macro-Parametric Approach". *Int. J. of CAD/CAM,* **2**(1), pp. 13–21.

[11] Li, M., Gao, S., and Wang, C., 2007. "Real-Time collaborative Design With Heterogeneous CAD Systems Based on Neutral Modeling Commands". *ASME J. Comp. Inf. Sci. Eng.,* **7**(2), pp. 113–125.

[12] Harper, R. Practical Foundations for Programming Languages. http://www.cs.cmu.edu/~rwh/plbook/book.pdf.

[13] Ridgway, J. Foundations for Polylingual Systems.

[14] Ridgway, J., and Wileden, J., 2004. "Toward Formal Foundations for Exceptionally Safe Interoperability Among Object-Oriented Languages". *Computer Science Technical Report*, Dec.

[15] Hirzel, M., and Grimm, R., 2007. "Jeannie: Granting Java Native Interface Developers Their Wishes". In OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object oriented programming systems and applications, ACM, pp. 19–38.

[16] Siméon, J., and Wadler, P., 2003. "The essence of xml". *SIGPLAN Not.,* **38**(1), pp. 1–13.

[17] Cheney, D., 2008. "3D CAD Model Validation". *3D Collaboration & Interoperability '08.*, May.

[18] Ullman, J., 1976. "An Algorithm for Subgraph Isomor-

phism". *Journal of the ACM,* **23**(1), pp. 31–42.

[19] McKay B.D. Department of Computer Science, Australian National University Canberra ACT 0200, A., 2006. Nauty User's Guide (Version 2.4).

[20] Cordella, L., Foggia, P., Sansone, C., and Vento, M., 2004. "A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs". *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **26**(10).

[21] Schmidt, D., and Druffel, L., 1976. "A Fast Backtracking Algorithm to Test Directed Graphs for Isomorphism Using Distance Matrices". *Journal of the ACM,* **23**(3), p. 433C445.

[22] Corneil, D., and Gotlieb, C., 1970. "An efficient algorithm for graph isomorphism". *Journal of the ACM,* **17**(1), pp. 51–64.

[23] Zhang, K., and Shasha, D., 1989. "Simple fast algorithms for the editing distance between trees and related problems". *SIAM J. Compute,* **18**(6), p. 1245C1262.

[24] Bhavsar, V., Boley, H., and Yang, L., 2004. "A Weighted-Tree Similarity Algorithm for Multi-Agent Systems in e-Business Environments". *Computational Intelligence,* **20**(4), pp. 584–602.

[25] Melnik, S., Molina, H., and Rahm, E., 2002. "Similarity flooding: A versatile graph matching algorithm". *Proceedings of Eighteenth International Conference on Data Engineering*, pp. 117–128.

[26] Resource Description Framework (RDF). `http://www.w3.org/RDF/`.