A Semantic Feature Model in Concurrent Engineering

Yong-Jin Liu, Kam-Lung Lai, Gang Dai, and Matthew Ming-Fai Yuen

***Abstract*—Concurrent engineering (CE) is a methodology applied to product lifecycle development so that high quality, well designed products can be provided at lower prices and in less time. Many research works have been proposed for efficiently modeling of different domains in CE. However, an integration of these works with consistent data flow is absent and still in great demand in industry. In this paper, we present a generic integration framework with a semantic feature model for knowledge representation and reasoning across domains in CE. An implementation of the proposed semantic feature model is presented to demonstrate its advantage in knowledge representation by feature transformation across domains in CE.**

***Note to Practitioners*—In this paper, an important problem in manufacturing industry is addressed: how semantic features can be used across different domains in CE. The proposed hierarchical feature model in this paper offers a solution to two aspects in this problem: (a) how to capture the specific knowledge consistently in different domains and (b) how to reuse the existing data. The presented modeling approach is an attempt that uses a semantic feature language representation in CE and sheds some light on a practical solution to this difficult problem in industry.**

***Index Terms*—Concurrent engineering, knowledge representation, semantic features.**

I. Introduction

In the manufacturing industry, the development of products needs to go through all phases of product lifecycle [23]. The lifecycle includes design, manufacturing, assembly, inspection, maintenance, recycle, and disposal, etc. In the past 20 years, many methods have been proposed for different phases in product lifecycle. Several leading systems are computer-aided design (CAD), computer-aided process planning (CAPP), computer-integrated manufacturing (CIM), computer-aided inspection planning (CAIP), assembly sequence planning (ASP), product data management (PDM), and so on. These systems have been successfully applied in many industrial activities. However, these systems use inconsistent data representations and applying these systems in a sequential manner would make the process of product development unnecessary long and inefficient.

Concurrent engineering has been proposed in the mid of 1990s as a comprehensive philosophy in which simultaneous design of a product and all its related processes in lifecycle are taken into consideration in a parallel fashion [21], [23]. The methodology of concurrent engineering needs an efficient and unified knowledge representation across various disciplines which supports different action behaviors in a consistent way. An important application of a unified CE knowledge representation is to facilitate making decisions by a cross-functional product development team (CPDT). CPDT consists of representatives from different departments in a company. Communication between CPDT members is critical for product development in concurrent engineering. Since representatives in CPDT have different backgrounds, product descriptions with rich semantics are much better than the raw engineering data for communication. However, most commercial systems provide only rich engineering data but without semantic meanings. As reviewed in the next section, so far the research on semantic features in concurrent engineering achieves limited successes in this direction.

Humans can freely communicate with their common language. Motivated by the rapid development of natural language processing in artificial intelligence and its applications in machine communications [22], in this paper, we make an effort to model the multidiscipline data in CE using a semantic feature language representation. The proposed language representation is general and can serve as a unified framework that consistently integrates previous novel work in different CE domains, such as design by features [10], [14], process planning with manufacturing features [8], automated visual inspection [7], PDES/STEP-based integrated design and assembly planning [28], and so on. The inference in the language by feature transformation gives a practical solution to optimally searching and taking actions in various domains of product development.

II. Related Work

Feature modeling has been shown to be a promising tool in concurrent product design [21]. A feature is referred to as a higher level grouping of geometrical, topological and functional primitives into an entity more suitable for use in design, analysis or manufacturing [4]. One of its key advantages is allowing the designers to store a variety of heterogeneous information, such as material property, geometric and manufacturing information, tolerance and performance requirement, in a feature model with prior knowledge. Early feature models mainly concentrated on the geometric description [10], [19] and have been widely used in parametric, history-based CAD/CAM commercial systems.

To efficiently map between different product lifecycle phases, the detailed features need to be abstracted and integrated into a generic semantic model whose aspects are functionally significant. In a concurrent engineering environment, it is also required to design and modify different views of parts' features across different functionality domains, in an efficient and consistent way. Researches in this direction have attracted much attention and many works have been done. To use features in an abstract level, some work focused on feature validation and maintenance [3]. Chen and Hoffmann [5] studied the feature editability and reevaluation problem using persistent name matching techniques. Feature propagation, association and mapping across domains are studied in [9], [25]. Although different feature representation and manipulation techniques have been proposed, they are exclusively used in some specified scenarios. A general, integrated framework is still absent to characterize different techniques and make efficient communication among heterogeneous data in industry.

Knowledge representation and reasoning in complex domains have been studied in-depth in the field of artificial intelligence. Informally, artificial intelligence is to make machines behave like human beings. Semantic feature closely matches the way how human beings think about the product design and development. This motivates us to use feature language in concurrent engineering, in which usually intensive

Manuscript received July 05, 2009; revised September 21, 2009 and November 11, 2009; accepted December 27, 2009. Date of publication March 29, 2010; date of current version July 02, 2010. This paper was recommended for publication by Associate Editor M. Zhang and Editor V. Kumar upon evaluation of the reviewers' comments. This work was supported in part by the Natural Science Foundation of China under Projects 60736019, 60970099, the 863 Program of China under Project 2007AA01Z336, and the 973 Program of China under Project 2006CB303102.

Y. J. Liu is with the Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing, China (e-mail: liuyongjin@tsinghua.edu.cn).

D. Lai, G. Gai, and M. Yuen are with the Department of Mechanical Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2009.2039996

human intervention is required. Formal language [1] has proved to be a powerful tool for facilitating feature modeling. Using grammar has achieved some successes in automatic reasoning for product design and manufacturing [6], [24]. However, these language representation techniques have not been used in multidiscipline domains of concurrent engineering.

Recently, ontology-based approaches have been introduced into CE research. Patil *et al.* [20] proposed a product semantic representation language PSRL to exchange semantics data between SolidWorks and Unigraphics systems. PSRL is studied in domains including product design and computer-aided process planning. National Institute of Standards and Technology (NIST) develops a neutral ontology language called PSL [26]. PSL is mainly used to transform manufacturing process information between different software tools. In this paper, we propose a hierarchical semantic model facilitated with a language representation. A case study is depicted, showing an interesting application that extends the feature language representation to a broad range of domains in concurrent engineering.

III. Overview of Basic Framework

Denote by $\mathcal{X}$, a universal feature set in all phrases in concurrent engineering. $\mathcal{T}$ is a collection of subsets of $\mathcal{X}$ that satisfies the conditions:

1) The set $\mathcal{X}$ and empty set $\emptyset$ are in $\mathcal{T}$;
2) If $\mathcal{O}_1, \mathcal{O}_2 \in \mathcal{T}$, then $\mathcal{O}_1 \cap \mathcal{O}_2 \in \mathcal{T}$;
3) If $\mathcal{O}_1, \ldots, \mathcal{O}_k \in \mathcal{T}$, then $\mathcal{O}_1 \cup \cdots \cup \mathcal{O}_k \in \mathcal{T}$, $k \geq 2$.

$\mathcal{T}$ is called a topology on $\mathcal{X}$ and $(\mathcal{X}, \mathcal{T})$ forms a topological space [2]. The operation * on $\mathcal{T}$ assigns each ordered pair $\langle x, y \rangle$ of elements of $\mathcal{T}$ one element $x * y$ in $\mathcal{T}$. The set $\mathcal{T}$ along with operation $*$ forms an algebraic structure [17] if the following properties are obeyed:

1) $x * y$ is defined for every ordered pair $\langle x, y \rangle$ of elements of $\mathcal{T}$;
2) $x * y$ is uniquely defined;
3) $\mathcal{T}$ is closed under the operation *.

Akin to the geometric features defined in [10], an *Invalid* element is included in $\mathcal{T}$ so that

$$\forall x \in \mathcal{T},\ x * Invalid = Invalid * x = Invalid.$$

Four commonly used operators in algebra are union $(\cup)$, intersection $(\cap)$, subtraction $(-)$, and complement $(/)$. To facilitate semantic feature transformation using a language representation, we implement these operations in a much restricted manner. Section IV-B presents implementations of these restricted operations, i.e., equality $(=)$, grouping $(+^*)$, reference $(\neg)$, and comparison $(/-)$. These operators suffice to capture the operations considered in this paper.

A semantic model is presented in Section IV to define the complete feature set $\mathcal{T}$ in a consistent and hierarchical way. With the semantic model, a feature language is used as a neutral, intermediate representation for semantic intra and interoperability. One crux in feature based product modeling in CE is about feature transformation across domains. As an efficient solution that benefits from language description, a reasoning processing for feature transformation is presented in Section VI with the aid of a reasoning table. The vocabulary of the language includes:

1) Objects Obj that are either the product or parts in a product.
2) Features in $\mathcal{T}$.

The grammar is defined as $\mathrm{Gr} = (P, G_{\mathrm{obj}})$:

$$\begin{aligned} P : \{\ & (1)\ G_{\mathrm{obj}} \rightarrow F && : \text{intialize the feature graph} \\ & (2)\ F \rightarrow Ft|t && : \text{define the aggregate predicate string} \\ & (3)\ t \rightarrow F * F && : \text{define predicate string} \\ & (4)\ * \rightarrow \varphi_p, \forall \varphi_p \in * && : \text{instance} \\ & (5)\ F \rightarrow f_i, \forall f_i \in F && : \text{instance} \\ \} & \end{aligned}$$

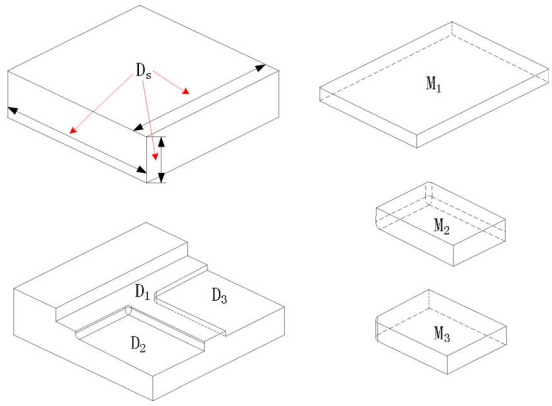

Fig. 1. Feature transformation in pocket manufacturing: From four design features (design feature D_s for stock and design features D_1, D_2, D_3 for three pockets, respectively) to three manufacturing features (M_1, M_2, M_3 for three pockets, respectively).

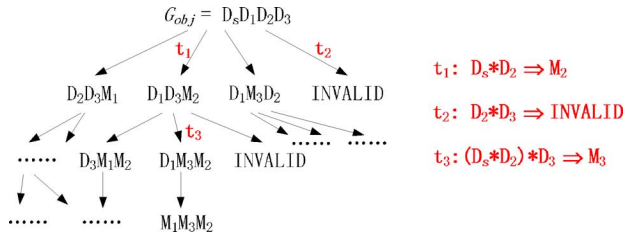

Fig. 2. String expansion in a feature based machining sequence generation.

where F is the feature graph of object G_{obj}, t is one of instances of the vocabulary class, | means "or" and * is the operation set $\{=, +*, \neg, /-\}$. In this language representation, feature transformation or reasoning is encoded in predicate string $t \rightarrow F * F$.

For a simple illustrative example, Fig. 1 shows a stock and a part. Four design features (D_s for stock and D_1, D_2, D_3 for three pockets in part, respectively) form the start symbol G_{obj}. In a feature based machining sequence generation, three manufacturing features (M_1, M_2, M_3 for three pockets, respectively) need to be generated in an optimal order. The task-driven string expansion is illustrated in Fig. 2. At first glance, the exhaustive expansion in Fig. 2 seems inefficient and the graph will grow exponentially. Actually high efficiency can be achieved by applying informed expansion with some search strategies. In the example of pocket manufacturing, cost can be assigned to each predicate, e.g., the machining cost to $D_s * D_2 \rightarrow M_2$ is estimated by removed material and used tools. By designing admissible heuristic cost functions using problem-specified knowledge [11], the well known A^* search efficiently prunes unnecessary subtrees from the graph [8] and the search for optimal solution can be done in linear time [16].

To apply the proposed feature language for knowledge representation and data reuse, the feature list of a product is stored in data repository of the system, e.g., $G_{\mathrm{obj}} = \{(D_s D_1 D_2 D_3)\}$ for the part in Fig. 1. If a particular view (manufacturing feature for an example) of product data is inquired, the feature type is investigated in G_{obj}. If the view does not exist, task-driven inference with reasoning table is performed. Then, the data $G_{\mathrm{obj}} = \{(D_s D_1 D_2 D_3), (M_1 M_2 M_3)\}$ is updated and stored. The feature model proposed in Section V establishes the relations and constraints among individual features. The following relationships can be retrieved from G_{obj}:

- D_s, D_1, D_2 and D_3 are pairwise dimensionally constrained;
- D_s infers M_1, M_2 and M_3;

- D_2 infers M_2 and M_3;
- M_2 is inferred by D_s and D_2.

When the product data $G_{\rm obj}$ is reused with modifications such as:

- D_2 is modified: (1) the constraints between (D_2, D_s) and (D_2, D_1) are examined to verify whether this modification is feasible or not and (2) the inferred feature M_2 is updated;
- M_2 is changed: the constraints in D_s and D_2 that infer M_2 are examined to verify whether this modification is feasible or not.

So the reused data is consistently updated in the repository.

To implement the feature language, a prototype concurrent environment that covers four typical domains including design, manufacturing, assembly and inspection, is considered in this paper.

IV. A Semantic Feature Model

In a CE environment, the semantic feature language is defined to be a language used to represent and communicate information within or across different domains, in terms of semantic features. The vocabulary of the semantic feature language consists of the basic semantic features, their children classes and five operators.

A. Semantic Features

A *semantic feature* is defined as a group of abstract data with attributes:

$$
\begin{array}{lll}
Semantic\_feature & ::= & \langle Abstract\_data\rangle\langle Attributes\rangle \\
 & & \langle Concurrent\_para\_list\rangle \\
Abstract\_data & ::= & \langle Feature\_name\rangle\langle Feature\_type\rangle \\
Attributes & ::= & \langle Static\_attributes\rangle \\
 & & \langle Dynamic\_attributes\rangle \\
Concurrent\_para & ::= & \langle Feature\_func\rangle\langle Mating\_feature\rangle \\
 & & \langle Mating\_component\rangle \\
 & & \langle Infer\_from\rangle\langle Infer\_to\rangle
\end{array}
$$

where

$$
\begin{array}{rcl}
::= & : & \text{means ``compose of''} \\
\langle\rangle & : & \text{means ``contents''}
\end{array}
$$

Other symbols used in this paper include

$$
\begin{array}{rcl}
\ll\gg & : & \text{means ``member of''} \\
[\,] & : & \text{means ``constraint'' or ``static attribute''.}
\end{array}
$$

The above defined semantic feature is in an abstract class which coordinates derived classes of the hierarchical features at different levels and in different domains. The detailed descriptions of the abstract class members are as follows.

Abstract data includes the name of the semantic feature and its type which are used to identify corresponding features in various domains. A semantic feature has two types of attributes: static and dynamic attributes. Static attributes are constraints and their values are not allowed to be modified. It is only valid within a specified domain and only affects the feature classes derived from it. Dynamic attribute values can be modified at any time within a reasonable range. When inherited in a derived class, a dynamic attribute can become a static attribute.

Concurrent parameters of the semantic feature describe the feature's function and its relationships with features of other components. In addition to setting up the constraints with other features, concurrent parameter also enables the semantic feature supporting the conceptual model of a product.

The data $\langle Infer\_from\rangle$ and $\langle Infer\_to\rangle$ in concurrent parameter establishes a graph structure in the ontology defined by the hierarchical semantic model. When the product data is reused with modification, the static attributes (constraints) of features in $\langle Infer\_from\rangle$ are checked for verification and the features in $\langle Infer\_to\rangle$ are updated.

B. Five Operators

1) Equality $(=)$*:* Two features A and B are said to be equal, written as $A = B$, if and only if A and B satisfy:

- represent the information in the same domain;
- are on the same detail level;
- all the corresponding abstract data and attributes have same data types;
- all the corresponding abstract data and attributes have same equal values.

If $A = B$ does not exist, then $A \neq B$. To satisfy the algebraic structure of feature set description, $A = B$ should return a composite feature in $\mathcal{T}$. So *FALSE* and *TRUE* are two features included in $\mathcal{X}$.

2) Grouping $(+^*)$*:* The grouping, $A +^* B$, of two feature classes combine all attributes of A and B, similar to the union operation in set theory. *Constraints* on the two operands are:

- $A \neq B$;
- The abstract data of A and B have the same data types and the same values;
- A and B cannot be simultaneously in the same domain and at the same detail level.

The constraints limit the operands such that grouping reflects information of the same feature either in different domains, or in the same domain but at different detail levels.

3) Reference $(\neg)$*:* The reference, $A\neg B$, of two feature classes, manipulates the first operand and retrieves information from the second operand when necessary. *Constraints* on the two operands are:

- $A \neq B$;
- The abstract data of A and B have the same data types and the same values.

The constraint limits the operands being the same object of features either in the same domain or in different domains. Copying a feature A is achieved by $A\neg\emptyset$.

4) Comparison $(/-)$*:* The comparison, $A/-B$, of two features A and B identifies the differences between the corresponding dynamic attributes of A and B. *Constraints* on the two operands are:

1) Their abstract data has the same data types and values;
2) They belong to the same feature class and at the same detail level.

The comparison operator manipulates the dynamic attributes of the operands only.

5) Reasoning $(\Rightarrow)$*:* The reasoning, $A \Rightarrow B$, where A is a semantic feature or a composite feature as input and feature B is output, means that B is deduced from A based on available knowledge. The details of reasoning process by feature transformation is presented in Section VI.

V. Feature Taxonomy

From the most abstract semantic feature, the derived detail features at different levels enrich the vocabulary of the language.

A. Design Feature

The design feature class meets the design-by-feature requirements and carries the geometric information. A component in the design domain is described by a list of design features, a list of surfaces and material information. The surface list covers all surfaces enclosing the

volume of the component's shape. Material information includes material type, hardness and heat treatment methods. The syntax of design feature is:

$$
\begin{array}{lll}
Design\_feature & ::= & \ll Feature \gg \langle Geometry\rangle \\
 & & \langle Topology\rangle\langle Design\_quality\rangle \\
Geometry & ::= & \langle Position\rangle\langle Orientation\rangle \\
 & & \langle Dimension\rangle\langle Surf\_list\rangle \\
 & & \langle Geom\_Operator\rangle \\
Topology & ::= & \langle B\_rep\_object\rangle \\
Design\_quality & ::= & \langle Geom\_tolerance\rangle\langle Dimen\_tolerance\rangle \\
 & & \langle Surf\_finishing\rangle \\
Geom\_Operator & ::= & \langle Difference \| Union \| Intersection \\
 & & Topology\_check \| NULL\rangle
\end{array}
$$

The geometry of a design feature covers its position, orientation, dimensions, geometric operators and surfaces that form the solid volume. The topology of a design feature is the B-rep object provided by a solid modeler [18].

The design quality is composed of dimensional tolerances, geometric tolerances and surface finishing. Surface finishing is denoted by a structure that contains the finishing value and a pointer to the surface. For each dimension in the geometry of a design feature, the dimensional tolerance is represented by a tolerance range whose upper and lower boundary values are specified explicitly.

B. Manufacturing Feature

Manufacturing processes are normally divided into three sublevels: primary, secondary and finishing. The primary process is for stock production such as sand casting, forming such as rolling, injection molding, joining such as welding and riveting, surface treatment, heat treatment, etc. The secondary process covers machining processes. The finishing process is related to cleaning, coating, plating and other surface treatment processes. Manufacturing processes at each level carry their own feature lists.

$Manuf\_feature$ is a derived class of the abstract semantic feature and is a base class in the manufacturing domain:

$$Manuf\_feature ::= \ll Feature \gg [Prim \| Sec \| Finish]$$

$Primary\_feature$ is a derived class of $Manuf\_feature$, which contains semantic information of stock generation:

$$
\begin{array}{lll}
Prim\_feature & ::= & \ll Manuf\_feature \gg [Primary] \\
 & & \langle Primary\_para\_list\rangle \\
Primary\_para & ::= & \langle Tool\_name\rangle\langle Gen\_method\rangle \\
 & & \langle Setup\_data\rangle \\
Gen\_method & ::= & [Cast \| Form \| Inject\_mold \| Join \| \\
 & & Surf\_treat \| Heat\_treat \| Others] \\
Setup\_data & ::= & \langle Temperature\rangle\langle Pressure\rangle \\
 & & \langle Hold\_time\rangle\langle Others\rangle
\end{array}
$$

$Secondary\_manuf\_feature$ concerns machining processes:

$$
\begin{array}{lll}
Sec\_m\_feature & ::= & \ll Manuf\_feature \gg [Secondary] \\
 & & \langle Sec\_para\_list\rangle \\
Sec\_para & ::= & \langle Tool\_name\rangle\langle Setup\_data\rangle \\
 & & \langle Machining\_operator\rangle \\
Setup\_data & ::= & \langle Power\rangle\langle Spindle\_speed\rangle \\
 & & \langle Feed\_speed\rangle\langle Cut\_depth\rangle \\
 & & \langle Setup\_no\rangle\langle Tool\_access\_direction\rangle \\
 & & \langle Feed\_direction\rangle\langle Fixture\_force\rangle \\
 & & \langle Others\rangle \\
Machining\_oper & ::= & [Mill \| Plane \| Drill \| Bore \| Ream \| \\
 & & Grind \| Others]
\end{array}
$$

$Finishing\_manuf\_feature$ concerns finishing process:

$$
\begin{array}{lll}
Fininsh\_feature & ::= & \ll Manuf\_feature \gg [Finishing] \\
 & & \langle Finish\_para\_list\rangle \\
Finish\_para & ::= & \langle Tool\_name\rangle\langle Operat\_method\rangle \\
 & & \langle Setup\_data\rangle \\
Operat\_method & ::= & [Clean \| Coat \| Plate \| Others] \\
Setup\_data & ::= & \langle Speed\rangle\langle Color\rangle \\
 & & \langle Paint\_thickness\rangle\langle Others\rangle
\end{array}
$$

C. Assembly Feature

In assembly domain, a component is described by a list of assembly features containing information on mating relations with other components attached to it. Assembly feature is a derived class from the abstract semantic feature and includes all information needed for assembly, such as the tools used for assembly, mating location, mating direction, assembly operations and assembly sequence, etc. The syntax of assembly feature is:

$$
\begin{array}{lll}
Assembly\_feature & ::= & \ll Feature \gg \langle Mating\_relation\_list\rangle \\
Mating\_relation & ::= & \langle Tool\_name\rangle\langle Setup\_data\rangle \\
 & & \langle Assembly\_operator\rangle \\
Setup\_data & ::= & \langle Mating\_location\rangle \\
 & & \langle Mating\_direct\rangle\langle Mating\_surf\rangle \\
 & & \langle Assembly\_sequence\rangle \\
 & & \langle Mating\_surf\_on\_other\_feature\rangle \\
 & & \langle Feature\_of\_other\_component\rangle \\
 & & \langle Assembly\_component\rangle \\
Assembly\_operator & ::= & [Insert \| Place \| Fasten \| Others]
\end{array}
$$

D. Inspection Feature

The inspection strategy can be classified into online and offline inspection. Online inspection, which mostly happens in the manufacturing secondary process, monitors the machining quality during cutting. Offline inspection is mainly the process of measuring dimensions, tolerances and surface finishing of workpiece, after each manufacturing setup.

In inspection domain, a component is described by a list of inspection features containing all criteria for quality control during and after the manufacturing process. Similar to other domain specified features, inspection feature is a derived class of the abstract feature including all information related to inspection, such as tooling, inspection operation, machining setups between which inspection is undertaken, standard followed, reference and target values, etc. The syntax of inspection feature is

$$
\begin{array}{lll}
Inspect\_feature & ::= & \ll Feature \gg \langle Inspect\_para\_list\rangle \\
Inspect\_para & ::= & \langle Tool\_name\rangle\langle Setup\_data\rangle \\
 & & \langle Inspect\_operator\rangle \\
Setup\_data & ::= & \langle Online\_flag\rangle\langle Ref\_feature\rangle \\
 & & \langle Machining\_setup\_no\rangle\langle Standard\rangle \\
 & & \langle Measuring\_type\rangle\langle Target\_value\rangle \\
Inspect\_operator & ::= & [Gauge \| Remote\_sensing \| Others]
\end{array}
$$

E. Feature Hierarchy

With object-oriented approaches, more detailed features at different levels and in different domains can be derived. We use the design feature as an example to derive more subtle children features. If form features defined in PDES/STEP are used for design features, children

TABLE I
PART OF A REASONING TABLE ON THE OPERATION $\mathcal{I} \Rightarrow \mathcal{O}$ WITH THREE ENTRIES WHICH ENCODE THE METHODS IN [10] AND [28], RESPECTIVELY

	Input $\mathcal{I}$	Output $\mathcal{O}$
1	$Design\_feature\_1$	$Design\_feature\_2$
2	$Design\_feature\_1/-$ $Design\_feature\_2$	$Sec\_m\_feature\_1$
3	$Design\_feature\_1\neg$ $(Sec\_m\_feature\_1+^*$ $Design\_feature\_2)$	$Assembly\_feature\_1$
...	... ...	...

design features, i.e., additive feature, subtractive feature, protrusion, primitive, join, depression and passage can be defined as:

$$
\begin{aligned}
Addition &::= \ll Design\_feature \gg [Union \| NULL] \\
Subtraction &::= \ll Design\_feature \gg [Difference] \\
Protrusion &::= \ll Addition \gg \langle Exit\_face \rangle [\rangle] \\
Primitive &::= \ll Addition \gg [NULL] \\
Join &::= \ll Addition \gg \langle Entry\_face \rangle \\
&\quad \langle Exit\_face \rangle [Union] \\
Depression &::= \ll Subtraction \gg \langle Entry\_face \rangle \\
&\quad [Difference] \\
Passage &::= \ll Subtraction \gg \langle Entry\_face \rangle \\
&\quad \langle Exit\_face \rangle [Difference]
\end{aligned}
$$

VI. REASONING IN THE FEATURE LANGUAGE

The reasoning process in the proposed feature language is defined by feature transformation. A semantic feature transformation describes a process that derives unknown attributes of a feature in a domain from the known features in the same or different domains, through proper knowledge. The knowledge is dependent on available resources in a particular factory and is encoded in a reasoning table: an example is shown in Table I. The feature transformation has the following rules:

- A feature transformation consists of both semantic features and operators.
- Semantic features are connected by operators in feature-operator-feature syntax like $A * B$.
- Reasoning is depicted by the operator $\Rightarrow$.
- Operations in a feature transformation start from left and end at right in a horizontal representation.
- The priority of operators from highest to lowest is $+*, /-, \neg, =, \Rightarrow$.
- Operators enclosed in parentheses have higher priority than the unclosed part.
- Each feature transformation has one and only one reasoning operator.
  - — The reasoning operator is the last operator in a feature transformation.
  - — The operand of reasoning operator can be a single feature or a composition of features with other operators.
  - — The output of reasoning operator is a single feature.

Semantic feature transformation encapsulates various functional activities across different domains in CE. The procedure of a task-driven feature transformation is as follows.

Input: A product data G_{obj}, a task and a reasoning table
Output: The updated data G_{obj} containing the required task feature
Begin

1. *Retrieve the $\langle Feature\_type \rangle$ of feature list in G_{obj}.*
2. *While (the task is not done) do (A^* search)*
   - *2.1. String expansion by $F \rightarrow Ft|t$*
   - *2.2. String absorption by reasoning $t \rightarrow F * F$;*
   - *2.3. Update data $\langle Infer\_from \rangle$ and $\langle Infer\_to \rangle$ according to the reasoning;*
3. *End while*
4. *Establish the relationship by updating the data $\langle Infer\_from \rangle$ and $\langle Infer\_to \rangle$ in both start features and task features.*
5. *Update product data G_{obj} by incorporating the task features.*
6. *End*

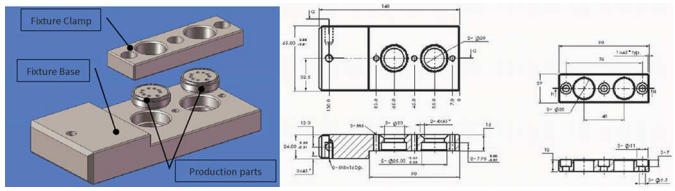

Fig. 3. An assembly model and its engineering drawing of metal block fixture, consisting of a fixture base and a fixture clamp.

VII. IMPLEMENTATION

The reasoning operator is implemented as a virtual function in the hierarchical feature classes, which will activate different knowledge bases and conduct different inference processes by checking the reasoning table. Various techniques, such as design for X (X stands for manufacturing [8], [13], production [10], fixture [27], assembly [28], inspection [7], recycling/disposal [12], optimal scheduling [15], etc.) techniques, as well as factory-floor-specified resources, can be implemented and generated into the reasoning table.

A proof-of-concept implementation of the proposed semantic feature model is presented. Fig. 3 shows an assembly model of a metal block fixture, which consists of a fixture base and a fixture clamp. The design features of fixture base and clamp are listed in Table II and illustrated in Fig. 4. Given the design features of fixture base and clamp, i.e., d_f_base and d_f_clamp, the assembly feature of fixture product is under investigation. In this case, the depth of the expanded tree rooted at $G_{obj} = (d\_f\_base, d\_f\_clamp)$ is 3:

$$d\_f\_base \Rightarrow d\_f\_base\_stock \tag{1}$$

$$d\_f\_base/-d\_f\_base\_stock \Rightarrow Sec\_m\_feature\_base \tag{2}$$

$$d\_f\_base\neg(Sec\_m\_feature\_base +^* d\_f\_clamp) \Rightarrow Assembly\_feature\_base \tag{3}$$

Reasoning operators (1), (2), and (3) are performed by invoking the corresponding operations in Table I. For reasoning 1,

$$Design\_feature\_1 \Rightarrow Design\_feature\_2$$

its attributes' modifications are[1]:

a) $Surf\_list(d\_f\_1)\neg Design\_quality(d\_f\_1) \Rightarrow Surf\_list(d\_f\_2)$.
b) $B\_rep(d\_f\_1) +^* Surf\_list(d\_f\_2) \Rightarrow B\_rep(d\_f\_2)$.

For reasoning 2 in Table I

$$Design\_feature\_1/-Design\_feature\_2 \Rightarrow Sec\_m\_feature\_1$$

the attribute modifications are:

a) $B\_rep(d\_f\_1/-d\_f\_2) +^* dimension(d\_f\_1/-d\_f\_2) +^* dimen\_tolerance(d\_f\_1/-d\_f\_2) +^* surf\_finishing(d\_f\_1/-d\_f\_2) \Rightarrow [CAST \| machined]$ $machining\_operator +^* cut\_depth +^* cut\_range +^* spindle\_speed +^* feed\_speed +^* cutter\_name +^* power$

[1] d_f_x is used for $Design\_feature\_x$

TABLE II
DESIGN FEATURE LIST OF THE FIXTURE IN FIG. 3

No	Base design feature	Clamp design feature
1	Primitive box	Primitive box
2	Slot Ø6.5	Through Hole Ø6.5
3	Threaded hole M_6	Through Hole Ø6.5
4	Threaded hole M_6	Through Hole Ø6.5
5	Threaded hole M_6	Boss
6	Threaded hole M_8	Boss
7	Through Hole Ø20	Boss
8	Through Hole Ø20	Through Hole Ø20
9	Packet	Through Hole Ø20
10	Packet	Chamfer
11	Chamfer	Chamfer
12	Chamfer	
13	Chamfer	

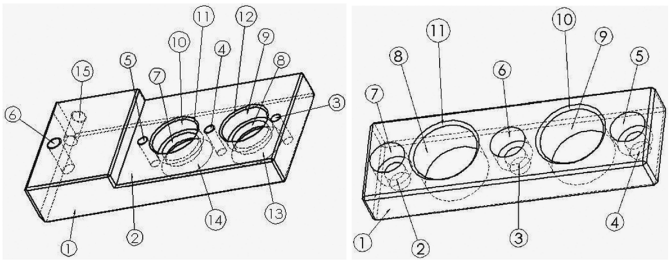

Fig. 4. The isotropic view of the fixture base and clamp with the design feature list in Table II.

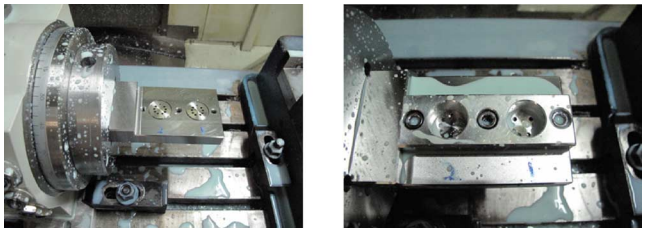

Fig. 5. The fixture product reasoning from the data in Fig. 3.

b) $Orientation(d\_f\_1) +^* machining\_operator \Rightarrow tool\_access\_direction +^* feed\_direction$
c) $machining\_operator +^* cutter\_name +^* power \Rightarrow tool\_name$
d) $Geom\_tolerance(d\_f\_1) +^* machining\_operator +^* tool\_access\_direction \Rightarrow setup\_no$

For reasoning 3 in Table I

$$d\_f\_1 \neg(Sec\_m\_feature\_1 +^* d\_f\_2) \Rightarrow Assembly\_feature\_1$$

the attribute modifications are:

a) $B\_rep(d\_f\_1) +^* dimension(d\_f\_1) +^* dimen\_torelance(d\_f\_1) +^* surf\_finishing(d\_f\_1) \neg geom\_tolerance(d\_f\_2) \Rightarrow mating\_surface +^* mating\_surf\_on\_other\_feature +^* feature\_of\_other\_component +^* assembly\_component +^* assembly\_operator$

b) $(d\_f\_1) +^* orientation(d\_f\_1) +^* assembly\_operator \neg(Position(d\_f\_1) +^* orientation(d\_f\_1)) \Rightarrow mating\_location +^* mating\_direction +^* setup\_no$

The snapshot of the final fixture product is illustrated in Fig. 5.

A. System Evaluation

Unigraphics Solutions UG NX5 is an integrated CAD/CAM/CAE commercial system. We present both the proposed method and UG NX5 solutions to a local company at Hong Kong. A cross-functional team (CPDT) is employed in the company to address the product life-cycle issues at the beginning of product development process.

The members in CPDT are interviewed. All engineers in CPDT have experience on product design using SolidWorks and AutoCAD. But they have not used UG NX before. The same task, fixture design, was presented to the interviewees with both the proposed method and UG NX5 solutions. The interviewees were asked to comment both solutions.

All interviewees commented that the UG NX5 solution is powerful. One manager further commented that there are mass engineering data output from UG NX5 and he needs extraneous time to figure out the meaning of these data by checking the software interfaces.

All interviewees also commented that the proposed method is good for product process development. One senior engineer further commented that a company-specified reasoning table is easy for generation and maintenance: different departments can contribute and maintain their own reasoning operations separately. As a comparison, if company-specified resources need to be considered in UG NX5, plug-in modules have to be developed by the third party for UG NX5.

This preliminary user study shows that the proposed method gives a unified knowledge representation and reasoning for CE with rich semantic information. It will benefit a cross-functional product development team (CPDT) that contains representatives from different departments and with different working experience.

B. Limitations of the Method and Future Work

As an attempt of introducing feature language representation into a unified knowledge representation problem in the comprehensive CE environment, there are still many works to do in the future. In the case study, the depth of the expanded tree is only three. In real-world situations, a huge ontological representation needs to be built for feature hierarchy. This also results in a large reasoning table for feature transformation. The expanded search tree may also have a large branching factor even if A^* search is applied. Consequently, more advanced searching techniques including state-of-the-art pruning should be applied to efficiently answer the inquiries.

VIII. CONCLUSION

In this paper, we describe semantic features in a language representation, which is defined across different domains in a concurrent engineering environment. The vocabulary of the language is spanned by semantic hierarchical features and five fundamental operators. The reasoning operations in the language is defined by the rules of feature transformation, together with a reasoning table which can be tailored for any company with limited and special factory resources. Using the proposed feature language, well defined semantics features can be efficiently induced across domains. A case study is presented to show the potential of the proposed method.

ACKNOWLEDGMENT

The authors thank the associate editor and reviewers for their constructive comments that help improve this paper.

REFERENCES

[1] R. R. Allen, K. L. Brown, and Joanne, *Learning Language Through Communication: A Functional Perspective*. Belmont, CA: Wadsworth Pub. Co., 1986.
[2] M. A. Armstrong, *Basic Topology*. New York: Springer-Verlag, 1997.
[3] R. Bidarra, "Validity Maintenance in Semantic Feature Modeling," Ph.D. dissertation, Delft University of Technology, Delft, The Netherlands, 1999.

[4] W. Bronsvoort and F. Jansen, "Feature modelling and conversion—key concepts to concurrent engineering," *Comput. Industry*, vol. 21, pp. 61–86, 1993.
[5] X. Chen and C. M. Hoffmann, "On editability of feature-based design," *Comput.-Aided Design*, vol. 27, no. 12, pp. 905–914, 1995.
[6] M. Flasinski, "Use of graph grammars for the description of mechanical parts," *Comput.-Aided Design*, vol. 27, no. 6, pp. 403–433, 1995.
[7] H. Garcia, J. Villalobos, and G. Runger, "An automated feature selection method for visual inspection systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 3, no. 4, pp. 394–406, Oct. 2006.
[8] J. Han, I. Han, E. Lee, and J. Yi, "Manufacturing feature recognition toward integration with process planning," *IEEE Trans. Syst., Man, Cybern., Part B: Cybern.*, vol. 31, no. 3, pp. 373–379, 2001.
[9] K. Jha and B. Gurumoorthy, "Automatic propagation of feature modification across domains," *Comput.-Aided Design*, vol. 32, no. 12, pp. 691–706, 2000.
[10] R. Karinthi and D. Nau, "An algebraic approach to feature interactions," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 4, pp. 469–481, 1992.
[11] B. Khoshnevis, J. Park, and D. Sormaz, "A cost-based system for concurrent part and process design," *Eng. Econ.*, vol. 40, no. 1, pp. 101–124, 1994.
[12] A. Kriwet, E. Zussman, and G. Seliger, "Systematic integration of design-for-recycling into product design," *Int. J. Prod. Econ.*, vol. 38, no. 1, pp. 15–22, 1995.
[13] Y. J. Liu, Y. K. Lai, and S. M. Hu, "Stripification of free-form surfaces with global error bounds for developable approximation," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 4, pp. 700–709, 2009.
[14] Y. Liu, S. Gao, and Y. Cao, "An efficient approach to interpreting rigorous tolerance semantics for complicated tolerance specification," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 4, pp. 670–684, 2009.
[15] M. Ni, P. B. Luh, and B. Moser, "An optimization-based approach for design project scheduling," *IEEE Trans. Autom. Sci. Eng.*, vol. 5, no. 3, pp. 394–406, 2008.
[16] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley, 1984.
[17] C. Pinter, *A Book of Abstract Algebra*. New York: McGraw-Hill, 1982.
[18] A. Requicha and R. Tilove, Mathematical foundations of constructive solid geometry: General topology of closed regular sets Univ. Rochester, Rochester, NY, Tech. Rep. TM-27a, 1978.
[19] J. R. Rossignac, "Issues on feature-based editing and interrogation of solid models," *Computers & Graphics*, vol. 14, no. 2, pp. 149–172, 1990.
[20] L. Patil, D. Dutta, and R. Sriram, "Ontology-based exchange of product data semantics," *IEEE Tran. Autom. Sci. Eng.*, vol. 2, no. 3, pp. 213–224, 2005.
[21] B. Prasad, *Concurrent Engineering Fundamentals*. Englewood Cliffs, NJ: Prentice-Hall, 1997.
[22] S. Russell and P. Norvig, *Artificial Intelligence*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 2002.
[23] A. Saaksvuori and A. Immonen, *Concurrent Engineering Fundamentals: Product Lifecycle Management*, 3rd ed. Berlin, Germany: Springer-Verlag, 2008.
[24] L. C. Schmidt and J. Cagan, "GGREADA: A graph grammar-based machine design algorithm," *Research in Engineering Design*, vol. 19, no. 4, pp. 195–213, 1997.
[25] S. Subramani and B. Gurumoorthy, "Maintaining associativity between form feature models," *Comput.-Aided Design*, vol. 37, no. 13, pp. 1319–1334.
[26] *The Process Specification Language,* Jun. 2009, [Online]. Available: http://www.mel.nist. gov/psl/
[27] Y. Xiong and X. Xiong, "Algebraic strcture and geometric interpretation of rigid complex fixture systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 4, no. 2, pp. 252–264, 2007.
[28] X. Zha and H. Du, "A PDES/STEP-based model and system for concurrent integrated design and assembly planning," *Comput.-Aided Design*, vol. 34, pp. 1087–1110, 2002.

Dynamics of WIP Regulation in Large Production Networks of Autonomous Work Systems

Neil A. Duffie and Leyuan Shi

***Abstract*—In this paper, dynamic behavior is compared for two methods of local work in progress (WIP) regulation in autonomous work systems in production networks. In one method, work systems do not share information regarding the expected physical flow of orders between them; in the other, order-flow information is shared to compensate for the variable dynamic effects of physical order-flow coupling. In both methods, the work systems adjust production rate with the objective of maintaining a desired amount of local WIP. A linear discrete-time dynamic model of the flow of orders between work systems is used, which promotes identification of fundamental properties such as characteristic times and damping. The results demonstrate the need for order-flow information sharing in establishing desired network dynamic behavior. Examples are used to illustrate behavior in the general case of omnidirectional order flows and the special case of unidirectional order flows.**

***Note to Practitioners*—In the type of production network analyzed in this paper, each work system autonomously adjusts its production rate with the objective of maintaining a desired amount of local work waiting to be processed. It is known that production networks can exhibit unfavorable dynamic behavior; for example, oscillation of inventory in supply chains as suppliers respond individually to variations in orders, leading to recommendations that supply chains should be globally rather than locally controlled. However, decentralized planning and control methods are an increasingly important alternative. Dynamic models are used in this paper to demonstrate the need for and benefits of order-flow information sharing between the work systems to compensate for variations in the structure of physical order flows in such networks. The goals are to avoid slow or oscillatory response to disturbances and to establish and maintain desired network dynamic properties, particularly when the structure of order flows in the network is omni-directional.**

***Index Terms*—Autonomy, distributed control, dynamic modeling, production systems.**

I. Introduction

Production networks can exhibit unfavorable dynamic behavior. An example is oscillation of inventory in supply chains as individual organizations respond individually to variations in orders, leading to recommendations that supply chains should be globally rather than locally controlled and that information sharing should be extensive [1], [2]. Unfortunately, it is difficult to make all of the information necessary for robust control available to a centralized planning and control entity, especially when there are a large number of work systems in a production network. It is now recognized that decentralized coordination can be provided by logistic processes implemented by autonomous entities that can be the logistic objects themselves [3]. Decentralized planning

Manuscript received January 02, 2009; revised July 15, 2009; accepted October 30, 2009. Date of publication February 02, 2010; date of current version July 02, 2010. This paper was recommended for publication by Associate Editor F. Chen and Editor Y. Narahari upon evaluation of the reviewers' comments. This work was supported in part by the U.S. National Science Foundation under Grant DMI-0646697 and in part by the German Research Foundation under Grant SFB 637/2-A6 andGrant Br 933/16-1.

N. A. Duffie is with the Department of Mechanical Engineering, University of Wisconsin–Madison, Madison, WI 53706 USA (e-mail: duffie@engr.wisc.edu).

L. Shi is with the Department of Industrial and Systems Engineering, University of Wisconsin–Madison, Madison, WI 53706 USA (e-mail: leyuan@engr.wisc.edu).

Digital Object Identifier 10.1109/TASE.2009.2036374