# Hybrid Feature-Object Based Part Design Data Modeling for Integrated Manufacturing Enterprise Systems Implementation

Stephen C. Shih
School of Information Systems and Applied Technologies
Southern Illinois University, Carbondale, Illinois 62901-6614, USA

Email: shihcs@siu.edu

## ABSTRACT

This research first explores the integration issues in the implementation of Computer Integrated Manufacturing (CIM). A modern CIM environment contains major aspects of manufacturing enterprise systems, including product development and design, process planning, production planning and control, etc. In order to automate product design and manufacturing in modern manufacturing systems, the integration of Computer Aided Design (CAD) and Computer Aided Process Planning (CAPP) is an underlying issue. The development of a proper interface between engineering design and manufacturing planning stages is at the center of CAD/CAPP integration. The representation or modeling of product design data is further identified as a key element for the successful implementation of design-planning interface development. Addressing the identified CIM integration issues, a new product design data model for the effective implementation of CIM is proposed. More precisely, this research adopts the object-oriented, feature-based product design data representation for seamless interfaces between the product design and subsequent manufacturing planning phases in Computer Integrated Manufacturing (CIM) systems.

**Keywords:** Computer Integrated Manufacturing, Computer Aided Design, Manufacturing enterprise systems integration, Automated part feature recognition and conversion

# INTRODUCTION

In essence, integrated manufacturing enterprise systems encompass a number of major functional areas: product development and design, process planning, material requirements planning, production planning and control, etc. Computer Integrated Manufacturing (CIM) involves the integration of those individual functional areas with the use of assorted computer-aided systems and enterprise-wide databases. Essentially, CIM can be conceived as a method or a philosophy of consolidating separate islands of automation of the entire production lifecycle into a computer-assisted distributed processing and control enterprise system (Groover, 2007). The part design and production data processed in a CIM system reveals a nature of heterogeneity and intricacy in terms of distributed information processing (Balakrishna et al., 2006**)**. Hence, communication among different automation islands in CIM poses a major challenge without a uniform part design interface. Specifically, one important aspect of CIM integration deals with the communication between part design and subsequent process and production planning tasks (Zhou, 2007). Engineering drawings are the medium conventionally used by the product designers to communicate with manufacturing engineers. Nonetheless, these drawings may sometimes contain insufficient or unstructured data that cannot be directly used in subsequent process and production planning tasks (Shah & Mäntylä, 1995). Meyer et al. (2009) stressed the importance of manufacturing integration for streamlining the production process and presented the integration concept from CIM to the digital factory. They give emphasis to the use of integrated manufacturing execution systems tools for dynamic mapping and modeling to maximum process capability and manufacturing intelligence. As a result, a proper representation and data modeling of part design is considered one of the critical success factors for effective implementation of manufacturing enterprise systems integration.

Ismail et al. (2004) developed a rule-based algorithm for the extraction of cylindrical based features from a neutral data format, namely STEP (for the Exchange of Product Model Data) file produced by any Computer Aided Design (CAD) systems. They explored two primary approaches to feature-based modeling for CAD/CAM integration: design-by-features and feature recognition. With the method of design-by-features, part models are defined directly by adding, subtracting, or manipulating features created as instances of predefined feature types. The second approach, feature recognition, deals with computationally recognizing features from conventional geometric models or from neutral data format such as IGES, STEP. Ismail et al. (2004) asserted that this approach circumvents the constraints of design-by-features by specifying features from preset component descriptions. Cicirello and Regli (2001) explored the approach to using machining features as an index-retrieval mechanism for solid models. Specifically, machining features extraction is implemented by mapping the solid model to a set of STEP machining features. Bhandarkar and Ragi (2000) developed feature extraction system employing the STEP file as input to define the geometry and topology of a part and in turn created STEP output file for downstream activities such as process planning, production scheduling, and material requirement planning. Gao et al (2004) studied the conversion algorithm for coaxial hole-series machining feature extraction for gear box components as well as discussed the planar-type machining features and non-geometrical attribute features. Han et al. (2001) presented the research findings of integrating feature recognition and process planning in the machining domain for CAD/CAM integration.

Rosemann et al. (2009) used ontology to as a modeling tool for representational analysis of business processes. They suggested that there is no well-accepted reference frameworks used to evaluate and compare the capabilities of different process modeling or representation techniques. They further presented a comparative summary of previous representational analyses of process modeling techniques over time and made suggestions as to where ontology-based representational analyses stands in terms of its limitations and future development. Brahe and Bordbar (2009) developed a methodology to support semi-automatic transformation of high-level business process models into final executable systems with the use of a set of domain-specific modeling languages explicitly designed for a given enterprise.

Addressing the challenges of CIM design-planning integration issues, the primary objective of this research is to propose a new approach to the design of a uniform parts description and data modeling scheme for seamless integration of design and manufacturing planning. In this paper, two essential properties in an appropriate design data representation are identified: data conformity and data completeness. Most of the current generation of geometric modeling systems have their own data structures and geometric data models to represent geometric entities such as points, lines and other primitives. However, in a multi-user environment, such as in a CIM environment, a uniform design data representation is imperative. With regard to the issue of data conformity, a specially designed part description language, called *Hybrid Feature-Object based Part Design Representation (HFO-PDR)*, is proposed in this research. With the use of HFO-PDR, the inconsistency and redundancy problems in parts representation can be effectively resolved. The second prerequisite of a well-designed part data representation schema is the data completeness. The part design representation schema of most of the currently used geometric modeling systems do not include essential technical information, such as tolerance, surface-finish, and material conditions, and thus cannot properly drive process and production planning systems which depend on such information (Case, 1994). The proposed HFO-PDR attempts to include not only the geometry data but also such technical information. In this way, a part can be described sufficiently; as a result, misinterpretation among separate production agents can be greatly eliminated.

# FRAMEWORK OF HFO-PDR

The proposed HFO-PDR is a hybrid data model incorporating conventional query processing, object-oriented database processing, and knowledge-based data processing capabilities for factual data and procedural knowledge manipulations (Lai, 1988). This new data model will provide a dynamic, logical representation and organization of the real-world CIM objects (entities), constraints on the objects, and relationships among the objects. In other words, the proposed new data model is capable of supporting an object-oriented representation for part design representation and CIM factual data, such as engineering design, machining, tooling, and process capability data, as well as a rule-based representation for procedural knowledge manipulations and inferential reasoning processes (e.g., the reasoning process for automated machine cell design). With HFO-PDR, design and manufacturing data can be converted into practical knowledge. Consequently, a better system of decision-making for production

management is made possible by effectively using HFO-PDR based knowledge.

As a prerequisite, initial part representation schema should be properly selected for the design data from a specific CAD modeling tool (e.g., AutoCAD). Subsequently, different feature types are to be defined properly in terms of the initial part representation. The definition of OR-DBS comprises the following facets:

1. Definition of CIM object schema
2. Definition of CIM production rule schema
3. Definition of integrated object/rule (OR) data model
4. Use the proposed object-oriented data mode to represent part design data stored in the CAD system for use by the subsequent process planning task.
   a) Composite 3-D block part design representation
   b) Feature-based part design representation
   c) Object-oriented part design representation

**Definition of CIM Object Schema**

As mentioned above, HFO-PDR reflects the flexibility and strength inherent in object-oriented programming languages, knowledge based systems, and the relational database languages. In other words, HFO-PDR is consistent with message-passing paradigm of object-oriented systems and knowledge reasoning capabilities. In addition, the proposed data model holds the same features as those in conventional relational database systems: a data-definition language, data-manipulation language, and a data-control language. For the design of HFO-PDR, the following core concepts of object-oriented data modeling should be taken into consideration:

- A CIM class describing the data structure and functionality of its instance.
- A CIM class inheriting all the data structure and functionality declared in its superclass.
- A CIM class specializing some of the data structure and functionality that are defined by its superclass.
- A CIM class served as an abstract class that contains no instances to abstract common features of several of its subclasses.
- Encapsulation combining data structure and functionality into its instances. (It hides internal aspects of objects from outside world and declares which features of an object are accessible.)

In summary, there are six principle steps of CIM object schema development:

1. Identify CIM classes
2. Define class attributes
3. Define CIM class hierarchy
4. Create objects (instances) from the defined CIM classes
5. Object initialization
6. Extend object representation

*Step 1 – Identify CIM classes.* A CIM system has a multiplicity of information (data and knowledge) requirements. In this research, the effort was made to classify CIM data and knowledge according to the system functional classification. Using the abstract data type concept, a CIM class represents a cluster or a collection of similar system objects that share same sets of design, planning, and/or other production attributes. An object is an *instance* of a class. The class incorporates the definition of the structure as well as operations of the abstract data types for CIM databases. A CIM system can be assorted into different levels of classes, such as part class, part family class, component class, part feature class, material class, machine class, tool class, and process class. The basic format of CIM classes is shown as follows:

1. Class name
2. Private, public, and protected features.
    a) Internal representation (Int-Rep): used to describe the structure and properties of the group of similar objects and provide unique values for each instance of a class.

    where

    *Int-Rep := <data member list> and*
    *data member (instance variables ) := attribute_name + attribute_type*

    b) External operators (E-Op) (methods): used to manipulate the instances (objects) of the class.
    *E-Op := <function member list> <target object><argument1, argument2, ...>*

Three example defined function members are shown as below:

GetProcessPlan(): Get the process plan report for a given part.
GetProductName(): Get the name of the given product.
GetBOM(): Get the bill of material report for a given part or product.

Member functions are the "methods" or procedures used to describe the behavior of the collected CIM objects, such as the invocation of knowledge sources, inferential reasoning, application program execution, or read/write database. The selector of a message selects the appropriate inferencing method from the methods associated with the target object. Selector can be a forward chaining, a backward chaining, or other inferencing strategies. In C++ computer programming language, the public interface is made up from the features that are listed after *public* keyword while the keyword *protected* is used for the second interface. The following shows three CIM class examples coded in C++:

```
class Part {
    char *part_name;          // name of the part
    char *part_number;        // part number
public:
    void setName (char *newName) {   // member function
    name = newPartName;
    } ;
    void GetProcessPlan()
        void print() {
        printf("The part name is: %s  \n",part_name) ;
        } ;
    } ;
```

```
class PartFamily { /
      char *pf_number ;          // part family number as string
      char *pf_description ;                  // part family description slot as string
      Part  *parts[52]// array of parts in class
public :
      Part *part ;      // part of the class
      void initialize(char *n, char *t) {
            number = n ;
            time = t ;
      } ;
};
class ProcssPlan {
      Char *surface_type ;
      Char *process ;
      Char *sub_proc ;
      Char *cutting_tool ;
public:
      void AddOp() ;
      void print() {
      printf ("Surface type: %s \n", surface_type) ;
      printf ("Process: %s \n", process) ;
      printf ("Sub process: %s \n", sub_proc) ;
      printf ("Cutting tool: %s \n", cutting_tool) ;
      } ;
} ;
```

*Step 2. Define the CIM class attributes.* An attribute is a characteristic which is associated with an object or a class. Each attribute possess a value, say attribute type. An individual object attributes is called a slot. Each object can possess one or more properties. The following shows the basic attribute expression:

*<attribute name:attribute type>*
where attribute type: integer, real, string, date, boolean

Example:

```
(Part(part_name:#string)
      (part_number:#string)
      (material:#string))
```

*Step 3. Define CIM Class Hierarchy.* The following presents a hierarchical data representation by the class concept for CIM:

$CLASS_i$:{$P_{ij}$}
  $SUBCLASS_{i1}$:{$P_{i1}$}
    $OBJECT_{i11}$:{$P_{i11}$}
      $SUBOBJECT_{i111}$:{$P_{i111}$}
      $SUBOBJECT_{i112}$:{$P_{i112}$}
      ...
    $OBJECT_{i12}$:{$P_{i12}$}
      $SUBOBJECT_{i121}$:{$P_{i121}$}
      $SUBOBJECT_{i122}$:{$P_{i122}$}

...
$SUBCLASS_{i2}$:$\{P_{i2}\}$
...
$CLASS_i$:$\{P_i\}$
...

Where $X$:$\{P\}$ - class (subclass, object, or subobject) X possesses a set of properties P.

$\{P_i\} = \{p_{i1},p_{i2},...,p_{im}\}$
$\{P_{i1}\} = \{p_{i11},p_{i12},...,p_{i1n}\}$
$\{P_{i11}\} = \{p_{i111},p_{i112},...,p_{i11o}\}$
$\{P_{i111}\} = \{p_{i1111},p_{i1112},...,p_{i111q}\}$
$\{P_{i112}\} = \{p_{i1121},p_{i1122},...,p_{i112r}\}$
$\{P_{i12}\} = \{p_{i121},p_{i122},...,p_{i12s}\}$
$\{P_{i121}\} = \{p_{i1211},p_{i1212},...,p_{i121t}\}$
$\{P_{i122}\} = \{p_{i1221},p_{i1222},...,p_{i122u}\}$
$\{P_j\} = \{p_{j1},p_{j2},...,p_{jv}\}$

$im \le i1n \le i11o \le i111q$
$im \le i1n \le i11o \le i112r$
$im \le i1n \le i12s \le i121t$
$im \le i1n \le i12s \le i122u$

To illustrate the class hierarchical structure, the following shows an example of two selected part classes for a part database of 60 sample parts that are grouped into 11 part families using the clustering algorithm.

*part{part_name, part_number, material, heat_treatment}*

*part_family_001:{rotational, bolt_screw_stud}*
*part_001:{G4, E2, E22, E25, E32, E34}*
*part_004:{G4, E2, E26}*
*part_006:{G4, E2, E22, E25, E32}*
*part_008:{G4, E2, E25}*
*part_011:{G4, E2, E26}*
*part_013:{G4, E2, E22, E25, E32}*
*part_020:{G4, E2, E25}*
*part_024:{G4, E2, E25}*
*part_027:{G4, E2, E22, E25, E32}*
*part_030:{G4, E2, E22, E25, E32}*
*part_037:{G4, E2, E25}*
*part_038:{G4, E2, E26}*
*part_042:{G4, E2, E25}*
*part_045:{G4, E2, E25}*
*part_049:{G4, E2, E25}*
*part_052:{G4, E2, E25}*
*part_053:{G4, E2, E11, E26}*
*part_057:{G4, E2, E25}*
*part_060:{G4, E2, E25}*

*part_family_002:{rotational, bolt_w_nut}*
*part_016:{G4, E3, E11, E26}*
*part_047:{G4, E3, E11, E26}*
*part_056:{G4, E3, E5, E26}*

Where G1 - G9: General shape features
E1 - E35: External MP features
I1 - I15: Internal MP features

H1 - H15: Hole features

Two types of class hierarchies are defined in the CIM system: single inheritance and multiple-inheritance. The sequence to define inheritance is to define the upper-level class and then the associated subclasses. These two types of inheritances are defined in C++ code.

```
class part_family_001{
      char *pf_number;
      char *pf_description ;
      Part  *parts[20]
public :
      Part *part ;
      void initialize(char *n, char *t) {
            number = n ;
            time = t ;
      } ;
};

// Single inheritance
class part_001: public part_family_001 {
      . . .
} ;

class part_042: public part_family_001 {
      . . . } ;

// Multiple inheritance
class part_016: public part, public part_family_002 {
      . . .
} ;
```

*Step 4. Create objects (instances) from the Defined CIM classes.* An object represents any entity involved in the CIM system. Objects (or entities) include tangible objects such as parts, products, materials, machines, tools, fixtures, facilities, reports, and documents as well as intangible objects like systems, databases, functions/processes, operations, cost, processing time, etc. An object also represents knowledge of product design and production planning being reasoned on by the production rules. In this paper, two methods are used to create objects from the defined classes. The first method is referred to *as Static Allocation* where object variables are defined by using class name as their type while the second method, Dynamic Allocation allocates the instances of classes dynamically during run-time instead of compile time by applying the C++ "new" operator. The "new" operator allocates enough memory to hold the newly created instance object and returns a pointer to the new instance object of the class.

*Step 5. Object initialization.* Constructor operator is used to initiate the objects. When an instance of a CIM class is created, constructors are automatically invoked. Constructors are function members of a class that have the same name as the class. Message passing is then applied in the main program for the invocation of a function member of an object. The statement is read as "sending the message FunctionMemberY to the object ObjectVarX". The main program here is served a message sender and ObjectVarX is the message receiver.

Step 6 *Extend Object Representation.* Three basic object-oriented semantic modeling concepts have been identified (Booch et al., 2007; Farrel, 2008): object-class: instance-of relationship; object-class: inheritance; and superclass-class-subclass: generalization relationship (IS-A relationship). In this research, the other two extended semantic data modeling concepts are extended for the special data processing requirements in CIM applications: *Composite Object* (Ma, 2005). This concept depicts a heterogeneous set of objects which form a data hierarchy. In other words, a composite object is collection of component objects from a number of different CIM classes. This is especially useful in representing any given part (part components, finished products, etc.) which may be composed of a number of subassemblies and each subassembly can further contain another set of subassemblies or subcomponents. From the data modeling point of view, this schema is effective in manipulating the proposed feature-based part definition language discussed in later chapter. The composite object reveals A-PART-OF relationship which is superimposed on the aggregation relationship between an object (e.g., assembly) and other objects it references (e.g., subassembly).

**CIM Object Processing Strategies Determination**

There are a number of algorithms used in this research for the control of CIM class/object processing: single inheritance and multiple inheritance, abstract data type, encapsulation, attribute-pattern matching, data recouping by inheritance, and object identity. First, three steps are involved for inheritance control: inheritance priority setup, inheritability setup, and inheritance strategy determination. Through inheritance, new classes are built on top of an existing less specialized hierarchy of classes, instead of redesigning everything from scratch. The new class can inherit both the structure (instance variables) and the behavior (functions, methods) from existing classes. From a modeling point of view, inheritance is a very natural and powerful way for information organization (Parsaye, 1989; Parsaye and Chignell, 1993). As a summary, the way to determine the inheritance priority value is conducted as follows:

```
Algorithm: {Inheritance Priority Determination}
begin
      case 1
            If the inheritance slot has a value
            then use the value as the inheritance priority.
      case 2
      If the inheritance slot does not have a value (or hasn't been declared) but an
                  inheritance number has been declared
            then use the number as the inheritance priority.
      case 3
       If the inheritance slot does not have a value (or hasn't been declared) and the
                  inheritance number has not been declared
            then use 0.1 as the default value.
end
```

In addition to setting inheritance priorities which determine how the slot will compete with other slots when the children or parent level want to inherit from it, the inheritability meta-slot also determines whether or not a slot can be inherited from at all.

**Algorithm {Object Inheritability}**

```
input: object-X{property-set-X: prop[X]}, object-Y{property-set-Y: prop[Y]}
notation:  X ⊙ Y: X inherit the property from Y
           X ⊗ Y: X does not inherit the property from Y
           X ← y: X inherit a value y from the object Y
           X ≠ y:X does not inherit a value y from the object Y
begin
    if object-Y is_a subobject of object-X and
    property-set-X = property-set-Y and
    prop(N,X)  ← v   // the value property N of the object-X is v
    prop(M,Y) ← u
    then
        prop(N,Y) ← v   //object-Y inherit a value from object-X
    and
    prop(M,X) ≠ u// The parent object object-X does not inherit a value from  object-Y
    and
        prop(X,-) ⊗ prop(Y,-)  //do not inherit a property up to a parent
    and
        prop(Y,-) ⊗ prop(X,-) //do not inherit a property down to a subobject
end
```

Example: To elucidate the object inheritability, assume that:

part_family_002 = object-X
part_047 = object-Y

If N = G4 and M = E11, then
prop(N,X) =  prop(G4, part_family_002) ← 1 and
prop(M,Y) = prop(E11, part_047) ← 1

where 1 stands for the presence of the feature N or M.

Hence

(i) prop(N,Y) = prop(G4, part_047) ← 1 (part_047 inherit the value of G4 from part_family_002)
(ii) prop(M,X) = prop(E11, part_family_002) ≠ 1 (part_family_002 does not inherit the value of E11 from part_047)

Encapsulation is one of the most beneficial concepts in object-oriented systems. Encapsulation combines data structure and functionality into objects. It also hides two internal aspects of objects from the system users and declares which features of an object are accessible: internal data structure and internal functionality. In other words, encapsulation keeps all the CIM static knowledge or data pertaining to the special design, planning or manufacturing application entity bundled together with all the functionality that applied to it.

To access a given CIM object, two ways are provided by sending messages: (a) from a client (another CIM object that sends out a message) and (b) from an heir (from the object's subobject). The class structure itself acts as a pointer to a set of objects, with data held by the property associated with each object. CIM classes thereby provide a way to search through lists of objects in order to identify which objects meet a specific rule condition. This is called attribute-pattern matching which is depicted as follows:

```
Algorithm: {Attribute Pattern Matching}
input:   conditionI, I = 1,2,...,N
         actionJ, J = 1,2,...,M
         conditionX := <operator><objectα.attribute_><valueγ> (X ∈ {1,2,..,N})
         actionY :=  <operator><objectα.attributeΔ><valueδ) (Y ∈ {1,2,..,M})
begin
        // run rule_evaluation
            do while I <= N
                evaluate conditionI
                if conditionI = .false. then
                    exit the program  // the rule is verified as false
                end_if
                next I
            end_do_while
                do while J <= M
                    execute actionJ
                (when J = Y then a message is sent from the conditionX and invoke all
                    classes with objectα; actionJ is then applied to all the invoked objects)
                    next J
                end_do_while
end
```

Example: Consider the following rule condition:

*(is parts.rm_feature shafting)*

which reads, "Are there any objects in the class 'parts' whose property 'rm_feature' (raw material feature) has the value 'shafting'?" This example illustrates an attribute-pattern matching condition. Objects in class "parts" all have the attribute "shafting".

To execute the rules evaluation, the system must have the appropriate data on which to base its conclusions. In some cases, if the values of slot incorporated in rule IF-BLOCK conditions are unknown, the system must first recoup the values to complete the evaluation. The proposed system supports two mechanisms to choose the desired data from related sources. Taking the following example for illustration, consider the following condition and assume that there is no current value for the attribute "material.k"

*(is part_1.material modeled-ABS)*

which means, "Does the attribute 'material' of the single object 'part1' have the value 'modeled-ABS'?" At this point, does the system find an answer? Assume that there is a value for the attribute "attribute1" at the level of the class (parts) to which the object (part_1) belongs. By using the inheritance method, the value of the object's attribute can be inherited from the class to which the object belongs.

Another powerful object control strategy is object identity (Khoshafian and Copeland, 1986). An entity is a handle which distinguishes one entity from the other. In the proposed system, every object in the CIM system is assigned an identity that will be permanently associated with the object, no matter how the object's structural or state transitions are changed. And the identity will be maintained across multiple CIM application programs or transaction instantiation. Methods used in this research to identify the CIM objects are:

1. Using meta variable which contains a complete object.
   Format: *ClassName ObjectName;*
   such as *Part part_001 ;* (this declaration in C++ creates a part_001 object.
   Variable part_001 is declared to contain the complete part object.

2. Using pointers to present external identification for identifying the object's addressability
   Format: *ClassName *pointer ;*
   Example:
   *Part *part_pointer;*
   *part_pointer = new Part ; // create a new object*
   *part_pointer -> function1() ; // send*

**Integrated Object-Rule Representation for CIM Procedural Knowledge**

To represent reasoning procedures, the OO-EDS adopts production rules (Luger, 2008). Production rules are knowledge structures which let the system perform certain inferential search operations, such as backward or forward chaining, etc. along reasoning paths. In brief, a production rule is a chunk of CIM knowledge representing a situation and its immediate consequences. In this research, the basic format of a rule is represented as the following LISP-liked expression:

```
rule (
    (RuleNumber RuleDescription)
    (if (ConditionList))
    (then (Hypothesis))
    (execute (ActionsList))
)

where

    ConditionList := ((Cond1)(Cond2)(Cond3)...)
    Cond1 := [<Operator> <ClassName.AttributeName> <Value>]
    [<Operator> <ObjectName.AttributeName> <Value>]
    ActionList := ((Action1)(Action2)(Action3)...
```

which means

*"**IF** statement is followed by a set of conditions, then statement is followed by a HYPOTHESIS or goal which becomes true when the conditions are satisfied, and EXECUTE by a set of actions to be undertaken as a result of a positive evaluation of the rule (the conditions). "*

CIM knowledge is further classified into different groups, such as the Process Planning Knowledge Source:

- Rules of machining processes selection
- Rules of machine tools selection
- Rules of tooling selection
- Rules of fixtures selection
- Rules of operation sequencing determination
- Rules of cutting conditions
- Rules of standard time
- Rules of cost consumption
- Rules of stock determination

Based on the normalization rules, relational database files are generated through the Relational DB Transformer, including Feature Database: Part DB, Feature DB, Operation DB, Machine Tool DB, etc. The following steps illustrate how those databases are generated as well as their interrelationships:

1) Data of the first portion of the HFO-PDR, part_heading information, are extracted and stored in the PART database (partial structure).
2) Feature data in the second portion of HFO-PDR are then converted to Binary Feature Code (BFC) according to the presence or absence of the features. BFC is the binary variable representation describes parts with binary coordinates based on the presence or absence of certain part feature, where value 1 stands for the presence of the feature while value 0 represents the absence of the feature. A complete PART database will contain both heading information and the BFC.
3) BFC is then used as a key express to link the PART database and the FEATURE database.
4) Next comes the determination of necessary operations required for each feature. Through the Manufacturing Process Expert System (MPES), sets of operations are identified for relative features. The following shows a set of selected example rules of MPES for selection of necessary operation, cutting tool, machine for a sample part based on the hole feature:

```
rule (
      (R301 mp_hole )
      (if (is <part.feature> drill))
      (then (hole))
      (execute
                  ( set <operation> drilling)
                  (set <cutting_tool> drill)
                  (set <machine> drill press)))
rule (
      (R101 mp_external )
      (if (is <part.feature> cylinder))
      (then (external_manufacturing_process))
      (execute
                  ( set <operation>  turning)
                  (set <cutting_tool> round_nose_turning_tool)
                  (set <machine> lathe)))
```

```
rule (
      (R122 mp_external )
      (if (is <part.feature> neck_round))
      (then (external_manufacturing_process))
      (execute
                  ( set <operation>  turning)
                  (set <cutting_tool> cut_off_tool)
                  (set <machine> lathe)))
rule (
      (R125 mp_external )
      (if (is <part.feature> thread))
      (then (external_manufacturing_process))
      (execute
                  ( set <operation>  turning_threading)
                  (set <cutting_tool> threading_tool)
                  (set <machine> lathe)))
```

5) By linking the FEATURE database and the OPERATION database with the key expression "OP_ID", we can find detailed information about each set of necessary operations, such as the operation description, required machine ID number, tool number, and standard time.
6) Similarly, detailed machine tool information is found using the key expression “Machine_ID” that links the pointer from the OPERATION database to the MACHINE TOOL database.

# CONCLUSIONS

One of the prerequisites for engineering design-production planning integration is automated process planning. Without effective design-planning communications, a genuine integrated manufacturing enterprise system can hardly become a reality. Specifically, to achieve seamless integration, an interface is indispensible between computer aided design (CAD) systems and other computer-based planning and control systems, such as computer aided process planning (CAPP) and production planning systems. To achieve seamless integration between product design and manufacturing planning, the part/product design data should be properly converted into useful production-related knowledge for the subsequent planning and manufacturing tasks. This paper presents one of the foremost dimensions of integration: CIM data and knowledge integration. This dimension of integration involves the design of the HFO-PDR by developing a new form of feature-based data model which incorporate conventional database systems as well as object-oriented and knowledge-based data processing capabilities for factual data and procedural knowledge manipulations. Serving as a uniform part representation schemata for a more efficient integration tool, HFO-PDR can be transformed into different heterogeneous data modes (such as relational data mode, network data mode, object-oriented data mode, etc.) used in a number of individual production agents (e.g., sales/marketing module, inventory management module, material requirements planning module, machine cell design, etc.) in CIM. With HFO-PDR, inconsistency problems in parts representation can be avoided. A modern manufacturing facility varies more and more from the traditional manufacturing facility. Moreover, subjective decisions in describe parts can also be avoidable. In addition, a part can be described sufficiently by this scheme. Therefore, mistakes and misinterpretation among separate production agents can be greatly eliminated. With all these capabilities, an

effective CIM implementation can be easily accomplished by greater integration and management of manufacturing information throughout the whole production life cycle.

# REFERENCES


Balakrishna, A., Babu, R.S., Rao, D.N., Raju, D.R., & Kolli, S. (2006). Integration of CAD/CAM/CAE in Product Development System Using STEP/XML. *Concurrent Engineering*, 14(2), 121-128.

Bhandarkar, M. P., & Nagi, R. (2000). STEP-based Feature Extraction from STEP Geometry for Agile Manufacturing. *Computers in Industry*, 41, 3-24.

Booch, G, Maksimchuk, R.A., Engel, M.W., Young, B.J., Conallen, J., & Houston, K.A. (2007). *Object-Oriented Analysis and Design with Applications* (3rd ed.), Addison-Wesley Professional.

Brahe, S. & Bordbar, S. (2009). A Methodology for Domain-Specific Business Process Modeling and Implementation. *International Journal of Business Process Integration and Management*, 4(1), 5 – 17.

Case, K., Gao, J. X., Gindy, N. N. Z. (1994). The Implementation of a Feature-based Component Representation for CAD/CAM Integration. *Journal of Engineering Manufacture*. 208 (B1), 71-80.

Cicirello, V., & Regli, W. C. (2001). Machining Feature-based Comparisons of Mechanical Parts, SMI-2001: *International Conference on Shape Modelling and Applications*, Genova, Italy.

Farrell, J. (2008). *Object-Oriented Programming Using C++*, Course Technology.

Gao, J, Zheng, D.T, & Gindy, N. (2004). Extraction of Machining Features for CAD/CAM Integration, *International Journal of Advanced Manufacturing Technology*, 24(7-8).

Groover, M.P. (2007). *Automation, Production Systems, & Computer-Integrated Manufacturing* (3rd Edition), Prentice-Hall.

Han, J. H. Kang M., & Choi, H. (2001). STEP-based Feature Recognition for Manufacturing Cost Optimisation", *Computer-Aided Design*, 33, 671-686.

Ismail, N., Osman, M.R., Tan, C.F., Wong, S.V., & Sulaiman, S. (2004). Extraction of Cylindrical Features from Neutral Data Format for CAD/CAM Integration. *International Journal of Engineering and Technology*, 1(2), 206-212.

Jami J. Shah, J.J. and Martti Mäntylä, M. (1995). *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*, Wiley-Interscience.

Khoshafian, S. & Copeland, G. (1986). Object Identity, *Proceedings of OOPSLA-86*, Portland, OR.

Lai, S.H. (1988). A Knowledge-Based System for CAD/CAM Integration. *International Conference on Computer Integrated Manufacturing*, 396-404.

Luger, G.F. (2008). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving* (6th ed.), Addison Wesley.

Ma, Z. (2005). *Advances in Fuzzy Object-Oriented Databases: Modeling and Applications*, Idea Group Publishing.

Meyer, H., Fuchs, F., and Thiel, K. (2009). *Manufacturing Execution Systems (MES): Optimal Design, Planning, and Deployment*, McGraw-Hill Professional.

Parsaye, K. (1989). *Intelligent Databases: Object-Oriented Deductive Hypermedia Technologies*, John Wiley & Sons.

Parsaye, K. & Chignell, M. (1993). *Intelligent Database Tools & Applications: Hyperinformation Access, Data Quality, Visualization, Automatic Discovery*, Wiley.

Rosemann, M., Recker, J., Green, P., & Indulska, M. (2009), Using Ontology for the Representational Analysis of Process Modeling Techniques. *International Journal of Business Process Integration and Management*, 4(4), 251-265.

Zhou, X., Qiu, Y., Hua, G., Wang, H., & Ruan, X. (2007). A Feasible Approach to the Integration of CAD and CAPP. *Computer-Aided Design*, 39(4), 324-338.

**Stephen C. Shih** is Professor and Interim Director of School of Information Systems and Applied Technologies at Southern Illinois University Carbondale, earned his Ph.D. from the Pennsylvania State University, University Park, Pennsylvania. His academic career includes faculty positions in information systems and applied technologies, information systems and decision sciences, and manufacturing engineering. Besides his academic record, he has six years of industrial experience with United Technologies Research Center, Lucent Technologies, and SAMPO Electronics. His research areas of proficiency include supply chain management, intelligent systems design, and e-enterprise security management. He has published articles in many refereed journals, such as Journal of Computer Information Systems, International Journal of Electronic Business, and IEEE Transactions. Dr. Shih is currently serving as an editorial board member for five international journals.