Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2001), *15,* 145–171. Printed in the USA.
Copyright © 2001 Cambridge University Press 0890-0604/01 $12.50

Mechanical systems and assemblies modeling using knowledge-intensive Petri nets formalisms

X.F. ZHA, AND H. DU
School of Mechanical and Production Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798

(RECEIVED January 13, 1999; ACCEPTED June 5, 2000)

Abstract

This paper presents a novel knowledge-based Petri net approach to mechanical systems and assemblies modeling within a design with objects environment. A new unified class of object-oriented knowledge Petri nets, which can incorporate a knowledge-based system with ordinary Petri nets, is defined and used for the unified representations of assembly design and modeling. The object knowledge Petri nets, as a graphical language and a new knowledge-based description scheme, can be used to express the qualitative and quantitative aspects of the assembly design and modeling process in an interactive and integrated way. The four-level hierarchy model is proposed and constructed in terms of function-behaviors, structures, geometries, and features. The function-behavior-structure description is built on more abstract concepts so that it can match well top-down design. The static and dynamic characteristics in the design of assembly can also be captured. With the help of fuzzy logic, the incomplete, imprecise knowledge and uncertainty in the design process can also be dealt with. Therefore, the hybrid design object model can incorporate product data model, top-down design process, and assembly process model using an object-oriented, knowledge-based, feature-based, parametric, and constraint-based modeling approach, and can provide a more accurate and more flexible representation. To verify and demonstrate the effective use of the proposed hybrid design object model, a prototype system has been developed. This research provides a knowledge-intensive framework for intelligent assembly design and modeling.

Keywords: Artificial Intelligence; Assembly Modeling; Computer-aided Design; Design for Assembly; Design with Object; Petri Net

1. INTRODUCTION

From a designer's point of view, a machine is a structural model from general to detail that reflects certain relationships on different levels. The organization of part mating information to characterize assembly properties is one of the major issues in product assembly modeling. A mechanical system is often treated as a composite object, called an assembly model, in which each composite link carries the *is-part-of* relationships of its elements and subsystems, that is, the mechanical system and subsystems are represented by an assembly model. Product design generally involves creating formal models of the parts and their assemblies. The design process can be viewed as the process of creating a representation of the underlying objects. This representation can then be analyzed to derive important characteristics of the design process. The scheme for representation of a design process should capture major aspects of the process in a precise and concise manner. It is useful to have a direct relationship between the graphical and analytical representations. This is particularly advantageous in the analysis, simplification, and verification of a large system. The representation should also be able to describe abstractions and refinements in the design process, and integrate the basic principles and concepts of system design, such as analysis, modularity, and so on. The graphical representation appears to have the above-mentioned characteristics. However, the main requirement for any proposed representation model must be abstract and flexible enough in nature to be helpful for designers to design products in a top-down manner.

Although advances have been made recently in commercial CAD systems, particularly those employing parametric, variational, and constraint-based modeling methodologies,

Reprint requests to: X.F. Zha, School of Mechanical and Production Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798. E-mail: mxfzha@ntu.edu.sg

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

existing product-modeling software is not easily used for the construction of mechanical assemblies or otherwise suitable for assembly design. This is because the approaches developed in these systems are not flexible enough. Representations in these systems are only based on concrete single components and their feature relations. Features are still limited to a lower level and the model is difficult to update. The designer's thinking model is hard to set up due to the lack of an integrated design object model within a concurrent intelligent CAD environment. The machine design is not considered as a process, and it is therefore hard for designers to use in the course of top-down design (Mantyla, 1990). From the above observations, the main requirement of the design object model is that the model must be abstract enough to be helpful for a designer to design in a top-down manner, in which the designer develops design concepts by breaking down the design into a number of logical units, which are known as functional units of assembly. Functional units can be realized by "means" or "functional carrier." While functional units are combined to provide the required functions of the assembly, their corresponding means can be assembled into a structure which can be formulated both qualitatively and quantitatively. To build an efficient assembly modeling and design environment, the following features are crucial for the universal design object model:

1. Support abstract descriptions about geometries and their connections not only on a whole product assembly level, but on the level of single-piece parts;
2. Be helpful for designers to design top-down, that is, from incomplete and brief to complete and detailed;
3. Be well-structured for ease of conversion into other analytical models;
4. Be able to provide a knowledge framework to capture the designer's intention;
5. Have an arbitrary level of structure, that is, no limitation for the number of levels of hierarchical decomposition in a design object;
6. Can be treated as a single manipulating unit for analysis, evaluation, and version control;
7. Carry is-part-of relationship with some subtle classification about its semantics supporting full manipulation of composite objects; and
8. Support STEP-based data exchange at assembly level.

Petri nets have been very useful in many domains for modeling and analyzing discrete event systems like communication systems, databases, computer architectures, office automation, flexible manufacturing systems, and so on (Javor, 1995; Looney, 1988; Peterson, 1981). It is well known that Petri nets are a very powerful means of modeling discrete event systems, and they are also easy to use and interpret. Petri nets can be used as a graphical language to express, define, or specify assembly tasks in an interactive and environmentally independent way (Thomas et al., 1996; Zha et al., 1998c). The formalisms, structures and behaviors offered by the Petri net allow the designer to manipulate the views of assembly and to verify the assembly tasks in different ways. Also, the ease of use and interpretation of Petri nets can reduce the costs of programming and reprogramming. The designer can then make use of Petri nets for knowledge representations to incrementally describe a mechanical system or an assembly and for consistency checking and property verification in the design process (Chan et al., 1990; Gary et al., 1991; Zha, 1999).

The objective of this paper is to explore a novel application of Petri nets in modeling mechanical systems or assemblies and their design processes. To avoid the combinatorial explosion that arises in the product and its assembly process modeling when the number of pieces or objects involved increases, a new unified class of object-oriented knowledge Petri nets, which can integrate a knowledge-based expert system with ordinary place-transition Petri nets, will be defined and used for the representation and integration of distributed design models. Knowledge-intensive Petri nets can be used as a task-level graphical language to express the qualitative and quantitative aspects of the assembly and its design process in an interactive and integrated way. The proposed scheme for mechanical systems and assemblies modeling is based on knowledge-intensive Petri nets to construct a four-level hierarchy model from function-behavior, structure, geometric, to feature. In the following sections, some notations and assumptions for representations, definitions, and analysis techniques of place-transition nets and knowledge Petri nets will be first presented, and then the representations for mechanical systems and assemblies with the integrated object model will be discussed in detail.

The organization of this paper is as follows. Section 2 reviews the related work; Section 3 defines the knowledge-intensive Petri nets; Section 4 describes the design with object scheme for assembly design and modeling; Section 5 gives the details of the hybrid design object model in terms of a hierarchy of function-behavior, structure, geometry, and feature; Section 6 discusses a global-local data scheme for the hybrid design object model including product data exchange at assembly level; Section 7 overviews a prototype intelligent design and modeling system for mechanical systems and assemblies using the hybrid design object model; Section 8 outlines discussions, concluding remarks, and future work required.

2. LITERATURE REVIEW

Research on modeling of mechanical systems and assemblies has been going on for more than one decade. Several mechanical assembly representational schemes have been proposed to describe part-mating relationships. These include location graph, virtual link (Lee & Gossard, 1985), constraint graph (Wolter, 1988), relational model graph

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

(Homem de Mello, 1989), feature mating operation graph (Huang & Lee, 1989), functional relationship graph (Roy & Liu, 1989), and part position and part relation network (Heenskerk & Van Luttervelt, 1989). The basic concept is to store assembly entities, either parts, subassemblies, or parts with assembly operations, as vertices in various types of graphs. The variety of relationships between assembly entities, such as connectivity, geometry, location, and functionality, are characterized in terms of joining edges between graph vertices. Representation of assemblies can also be established in terms of a high-level language (Liebermann & Wesley, 1977; Popplestone et al., 1978a, 1978b; Liu & Glaser 1985; Takase & Nakajima, 1985), in the assembly mating-feature based graph approach (Liu & Popplestone, 1989; Nieminen et al., 1989; Rimscha, 1989; Shah & Rogers, 1993).

Other studies attempt to provide a mathematical model to represent linkages, functional volumes, tolerances, and allowances in assembly (Rocheleau & Lee, 1987; Kim & Lee, 1989; Giacometti & Chang, 1990; Wilson & Rit, 1991; Wilson, 1992), and to apply common modeling methods, such as geometric modeling, feature modeling, and knowledge-based modeling, to model mechanical parts and products in current CAD systems (Krause et al., 1993; Shah & Mantyla, 1995). Solid modeling can be used to create three-dimensional (3D) geometric models of individual parts of the assembly (Delchambre, 1992). Much work has been carried out on the addition of tolerance of information, dimensional or geometric, to the part solid model. Solid modeling, features, and attribute relationships are the basis for more complete product definition. Details about the geometric modeling, knowledge-based modeling, feature-based modeling for mechanical systems, and assemblies were reviewed in Krause et al. (1993) and Zha et al. (1998b). The representations mentioned above are, however, only based on concrete single components and their feature relations, although subassembly hierarchy could be described to some extent. Due to the development background mostly oriented toward supporting process planning and assembling, most previously proposed systems emphasize describing a final designed product or databases. The systems do not regard machine design as a process. As a result, it is hard for designers to use them in the course of top-down design.

Just as Dixon et al. (1990) pointed out that assembly design systems should enable designers to design top-down, starting at a high level of abstraction, a structure is composed of three kinds of objects: components, terminals, and connections. Liu (1992) defined a structure which has two kinds of basic elements: parts and joints. Based on this model, a mechanical system can be considered as a group of related parts and/or subsystems (subassemblies) assembled by joints. A subsystem (subassembly) is also an assembly of parts and its member subsystems connected by joints. Therefore, it is not difficult to imagine that the main concerns of the designer in structural design are no more than how to design a functional component, and then how to connect it to the others already existing. This means that the functional carries in a structure are logically divided into two types: component and connector (Gui, 1993). The component is a named concrete object, which performs desired functions of the machine in possible behaviors through connecting to the other components, while the connector is a named abstract object corresponding to joints, constraints, or operations and functions between two components. Constraints are not provided to the relevant components until the components are mated together. Thus any machine can be viewed as consisting of two basic classes of objects: a class of components and a class of connectors. Functional relations between features of individual components in a complete assembly modeling can be easily handled. The component-connector model could allow for the possibility that the properties take some values in the form of a fuzzy set over a base range. Liu's model (component-joint model) is very similar to Gui's model (component-connector model), but is at a lower level.

Based on bond graph, fuzzy logic, and object-oriented knowledge representation, Gui and Mantyla (1994) set up a function-centered scheme that aims to bridge the gap between functional modeling and feature modeling. They classify the design process model as three distinct but related models, namely, functional model, device model, and process model. Functional model is used to study the required functions of the design and determines how the overall functions can be achieved as an aggregate of low-level subfunctions. Device model (physical) is used to study the device (i.e., product) structures that can implement the required function. Process model is used to study the technical performance, behavior, and producibility of the structure. The component-connector model with multigraph data structure which joins function and structure is used for representing the conceptual model of a product. To link the device model and the final geometries of device structure Gui and Mantyla (1994) developed the concept of feature link, which can be used for recording features in the sequence of the design process and supporting feature inheritance. Using these concepts and ideas, Gui and Mantyla (1994) implemented a "Δ" top-down design system which includes three modules: DesignPlanner for design task analysis, DesignConsultant for design knowledge representation and consultation, and DesignSketcher for feature modeling.

More recently, formal approaches to capturing the logical interdependency relationships among parts and features in a complete assembly modeling have received much attention. Mantripragada and Whitney (1999) proposed a systematic approach to assembly design and modeling using a datum flow chain. Datum flow chain is a concept that captures the fundamental structure of a computer-aided design system in a top-down design process, including the designer's strategy for constraining the parts kinematically and locating them accurately with respect to each other. It relates the datum logic explicitly to the product's key charac-

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

teristics, assembly sequences, and choice of mating features, and provides the information needed for tolerance analysis. Two types of assemblies are addressed: Type 1, where the assembly puts together parts at their prefabricated mating features, and Type 2, where the assembly process can incorporate in-process adjustments to redistribute variation. Based on the concept of the datum flow chain, they also proposed a state transition model of assembly and concepts from control theory to model variation propagation and control during assembly (Mantripragada & Whitney, 1998; Whitney et al., 1999).

However, the existing approaches and models cannot explicitly express the notions of concurrency, causality, and conflict. They are not easily used as a graphical language to express, define, or specify, and simulate assembly tasks in an interactive and integrated way. The formalisms offered by them do not allow the designer to manipulate the views of assembly and to verify the assembly tasks efficiently. The designer cannot then make better use of them for knowledge representations to incrementally describe a mechanical system or an assembly and for consistency checking and property verification in the design process.

3. DESIGN WITH OBJECTS SCHEME

3.1. Design process modeling with objects

The design object model in the traditional CAD system is represented as being only a purely geometric entity. The most recent understanding of feature-based representation for machine design is characterized by both function and form (feature). Along the designer's thinking pattern, the top-down design manner is a natural way in which the designer develops design concepts by breaking down the design into a number of logical units which are known as functional units of assembly. Functional units can be realized by "means" or "functional carrier." While functional units are combined to provide the required functions of the assembly, their corresponding means can be assembled into a structure which is formulated both qualitatively and quantitatively. For most qualified designers, the design process from the conceptual solution to the structural configuration is usually carried out through a so-called "structure thinking block," with respect to a "function thinking block." The design process includes concepts and solutions evolving. This means that the design object is actually a dynamic object, and is ever changing throughout the design process (Deng et al., 1998).

Information processing in product design is inherently model based because the design object is structural in type. Therefore, object-oriented programming languages are desirable for declarative knowledge representation, object-oriented concepts, and fuzzy logic. As such, the object orientation scheme is employed so that both calculating and reasoning work in design can be carried out. The hybrid design object model is, in fact, an attempt to set up a knowledge framework in such a way that it becomes possible to process various types of knowledge in a top-down design process. Note that procedural representation with conventional languages such as FORTRAN, C, and PASCAL is not dealt with here, though it is important for the descriptions of algorithms and their processing in design analysis.

3.2. Object-oriented knowledge representation

The proposed object-orientation scheme is based on a mixed representative method and object-oriented programming (OOP) techniques, and allows designers to look at the design problem as a collection of objects or subproblems linked together by rules. Thus it provides the designers with an expressive power to represent complex problems or information in an effective manner. If a designer can break the design problem into the form of well-defined clearly manipulatable chunks with their own self-containing information which is interrelated through a series of rules and constraints, then these problems can be easily solved.

The basic structure of an object-orientation scheme is described as a unit. The class of object and its instances are described by using the unit structure. The object-oriented unit is composed of four types of slots, which are the relation slot, the attribute slot, the method slot, and the rule slot. The relation slot is used for describing the static relation among objects or problems. With the help of relation slot and according to the relation of classification, the design object can be described as a hierarchical structure. The knowledge existing in a super-class can be shared by its classes and subclasses. The messages that control design process can be sent among all instances of objects. In addition, if needed, other kinds of relation slots can be defined, such as the relation slot of resolution, the relation slot of position, and the relation slot of assembly, and so on. The attribute slot is used for describing the static attributes of a design object, such as width, length, material, and so forth. The method slot is used for storing the methods of design, sending messages, performing procedural control, and numerical calculation. The rule slot is used for storing sets of productions rules. The production rules can be classified according to the differences among objects being treated and stored respectively in rule slots in the form of slot value.

3.3. Overall design-with-objects architecture

The central design process inherent in a design-with-objects scheme can be represented as the architecture (O'Grady & Liang, 1998), as shown in Figure 1, with five main types of objects involved: namely, design models (S), design objects (O), design algorithms (A), functions (requirements and constraints) (FRC), and the evaluation schema (E). Object operators can express the relationship between these objects: inheritance, import, and message passing. The architecture in Figure 1 shows how the particular instance of a design model, S_1^k, is obtained from the

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

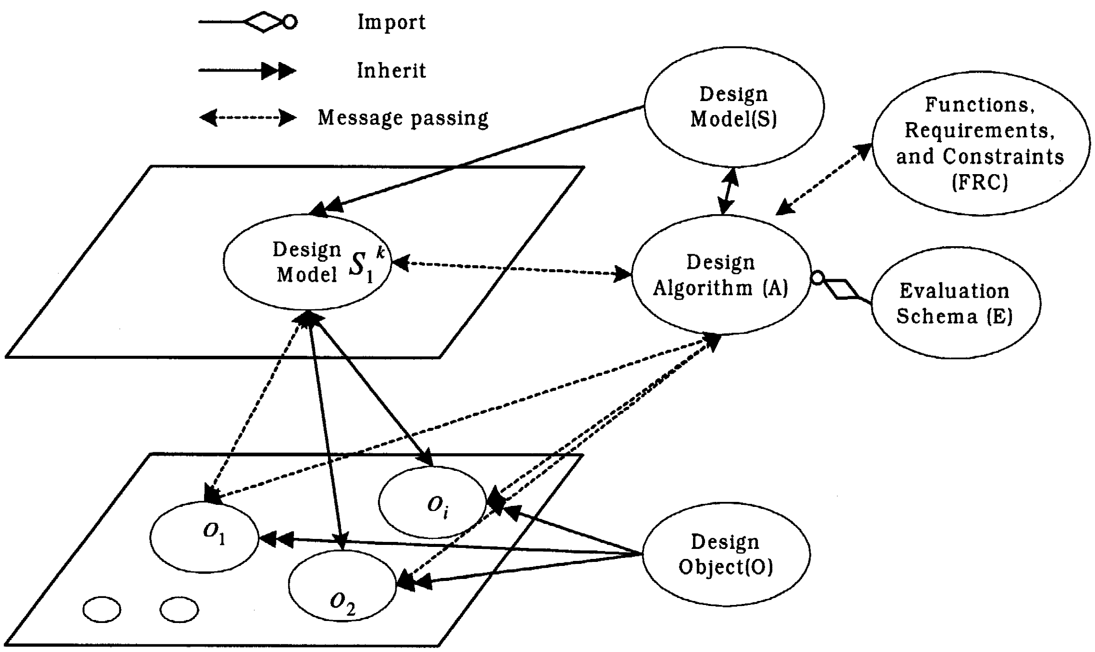


Fig. 1. The overall architecture of design with objects.

design algorithm, evaluation schema, requirements, constraints, and the design model object. For pure formulation design or creative design, a new design model object S is defined that describes the form of the model. A specific instance, S_1^k, of this design model can then be created. For pure parametric design then, the design model object S has already been defined and the design process therefore only involves the determination of a specific instance, S_1^k, of the design model. Note that additional objects can be defined within the overall architecture.

In the proposed design-with-objects scheme, the formalisms, structures, and behaviors offered by object knowledge Petri nets are used for the designer to model mechanical systems and assemblies from a function-behavior-structure description and to manipulate and verify the assembly design process. The details about knowledge Petri nets and knowledge Petri net-based design object model will be discussed below.

4. OBJECT KNOWLEDGE PETRI NETS

Since C.A. Petri first reported it in 1962, the Petri net has been analyzed, modified, and extended. Various classes of Petri nets have been built upon the extensions of the basic place-transition Petri nets. Petri nets possess the potential to be integrated into an artificial intelligence (AI) framework. For knowledge representation using Petri net, a new extension of Petri net called knowledge Petri net will be proposed.

4.1. Definition of knowledge Petri nets

A place-transition (P/T) net graph model, as shown in Figure 2a, can be defined as: $PTN = \{P, T, F, W\}$, where $P = (p_1, p_2, \ldots, p_m)$ is a place node set; $T = (t_1, t_2, \ldots, t_n)$ is a transition node set; and F is an arcs set which links between place nodes and transition nodes, and has the characteristics of: $P \cap T = \phi, F \subseteq (P \times T) \cup (T \times P)$, and $P \cup T = \phi$; and W: $F \to \{0,1\}$ is an association weight function on arcs, $\forall\ f \in F$, $W(f) = w_i$, w_i is the weight of arc f. Petri net is a directed place-transition net, as shown in Figure 2b. The net activities are based on a vision of tokens moving around an abstract network. Tokens are conceptual entities that model the objects and appear as small solid dots moving in a real network. A marked Petri net, as shown in Figure 2c, is formally defined as a 5-tuple $PN = (PTN, M_0) = (P, T, F, W, M_0)$, where PTN is a directed P/T net; P, T, W, F are the same as above definitions; M_0: $P \to \{0,1,2,\ldots\}$ is the initial marking. The Petri net graph is a graphic representation of Petri net structure and visualizes the reasoning rules. From a modeling perspective, these input and output places can represent the preconditions and postconditions of an event, or the resources required and released by an event.

In AI, a general problem can be expressed by a 3-element (X, f, R), where X is a set of the variables; $f()$ are the attributes of variables, expressed by the functions f: $X \to Y$, where Y is a multidimensional space; and R denotes the topologies on the set of the variables. From the Petri net notation above, the correspondences between the Petri net and the general description of a problem can be described as follows: $X \leftrightarrow \{P, T\}$, $R \leftrightarrow \{I, O\}$, $f \leftrightarrow \{C, M_0\}$. Thus, Petri net modeling can be considered to be equivalent to general problem solving strategies in AI. That is to say, a generic Petri net can incorporate the ordinary place/transition Petri net models into general problem description in artifi-

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

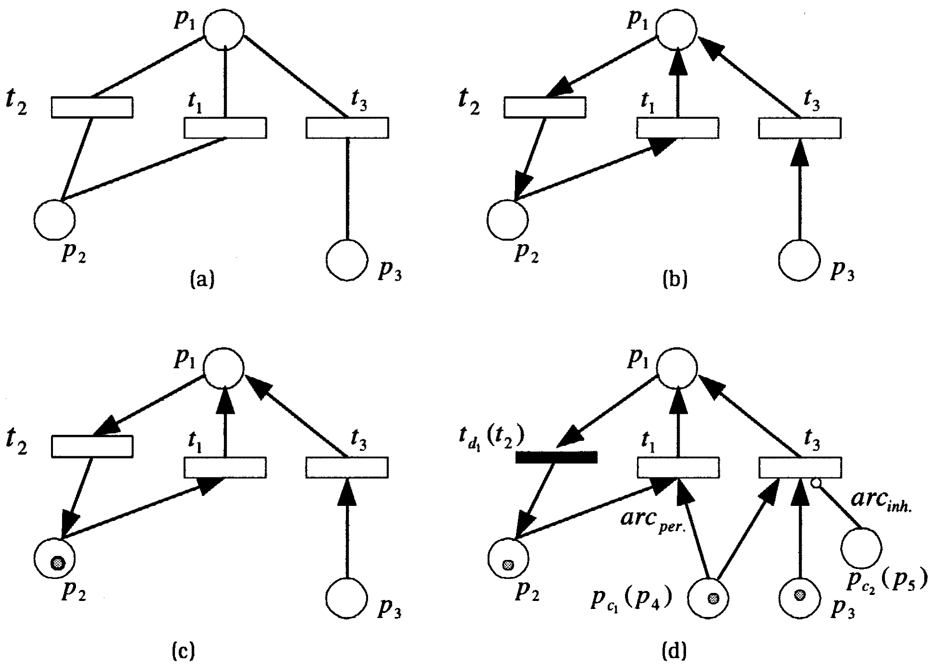


Fig. 2. Place-transition nets and Petri nets: (a) place-transition net, (b) directed place-transition net or Petri net, (c) marked Petri net, (d) knowledge Petri net.

cial intelligence. However, the major difficulty with ordinary Petri nets is that industrial applications are likely to result in large systems consisting of many places and transitions. Other shortcomings include the structural inflexibility and the inability to identify individual tokens. To use Petri nets for modeling complex systems, ordinary Petri nets should be extended with considerations of time, uncertainty, and knowledge information involved.

By incorporating the ordinary place/transition Petri net models into a knowledge-based expert system, a novel knowledge-embedded Petri net model can be defined as $KPN = (P, T, I, O, M_0, K)$, where (P, T, I, O) is a finite Petri net; K is the knowledge function defined from $P \times T$ into nonempty sets; $K(p)$ and $K(t)$ are the sets of knowledge associated with place $p \in P$ and transition $t \in T$, that is, $K(P) = \bigcup_{i=0}^{m} K(p_i)$, $K(T) = \bigcup_{j=0}^{n} K(t_j)$; $K = K_P \cup K_T$, where K_P: $P \rightarrow K(P)$ is the mapping from place set P to place knowledge set $K(P)$; K_T: $T \rightarrow K(T)$ is the mapping from transition set T to transition knowledge set $K(T)$. Therefore, the KPN model can be regarded as a combination of two aspects: net graph and knowledge annotations. Similar to an ordinary Petri net graph, a KPN graph is a graphic representation of knowledge-based Petri net structure and visualizes the reasoning rules. The annotations of knowledge are composed of the place knowledge annotations and the transition knowledge annotations, that is, K_P and K_T. The place knowledge is descriptive knowledge corresponding to the tokens and facts of the place in place set P. The transition knowledge is the rule knowledge (e.g., firing rules). Many methods for knowledge representation in AI such as proposition logic, attributes list, semantic network, frame, and If-Then production rule, and object orientation can be used to represent the knowledge in K_P and K_T.

By knowledge Petri net modeling, we mean that a knowledge Petri net model for a problem is first described as a kind of "template," and the model of the particular subproblems is then established as instances of the template. The correspondences between the knowledge Petri net and the general knowledge-based problem solving can be described as follows: $X \leftrightarrow \{P, T\}$, $R \leftrightarrow \{I, O\}$, $f \leftrightarrow \{K_P, K_T, M_0\}$. Thus, knowledge Petri net modeling is equivalent to general knowledge-based problem solving strategies in AI. As such, the knowledge annotations of KPN enlarge the representation ranges and contents of its net graph. These knowledge annotations can be organized into a knowledge base and an inference engine, and KPN is therefore a knowledge-based expert system. An illustration of a knowledge Petri net graph is shown in Figure 2d, where $P = ((p_1, p_2, p_3), (p_{c1}, p_{c2}))$; $T = ((t_1, t_3), (t_{d1}))$; $A = (\alpha, \beta) = (((p_1, t_{d1}), (p_2, t_1), (p_3, t_3), (p_{c1}, t_1), (p_{c1}, t_3), (p_{c2}, t_3)), ((t_1, p_1), (t_{d1}, p_2), (t_3, p_1)))$; $M_0 = (0,1,1,1,0)^T$; C_f: (p_{c1}, p_1), (p_{c1}, t_3) are permitted arcs, marked $arc_{per.}$, (p_{c2}, t_3) are inhibited arcs, marked $arc_{inh.}$; $W = (1,0,1,1, 1,1,1,1,1)$.

4.2. Object knowledge annotation

Various forms of knowledge Petri nets can be obtained by further extending or modifying the places or transitions of the knowledge Petri net described above. For instance, some extensions of places and transitions are used for function-

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

behavior-structure and feature-based geometric modeling for mechanical systems and assemblies. The definitions of these extended forms of knowledge Petri nets can be derived from object-oriented concepts and techniques. A template for knowledge annotations in knowledge Petri nets can be described as follows:

Knowledge entity name: ⟨name: string expression⟩ /* name of this entity */
Membership: ⟨place,transition⟩ /* the class types of this entity */
Superclass: ⟨name: string expression⟩ /*the parent entity class name */
Subclass: ⟨name: string expression⟩ /* the children entity class name */
Inherit method: ⟨overall, hidden, partial⟩ /* the inherit method of this entity class inherit knowledge from its parent class */
Method list: method name /* it is the port for other object to access */
Slot type: method
Value: { /* It contains the operation and activity when the method is invoked */
  filter (keywords); /* a filter to check the message */
  meta knowledge { /* meta knowledge is part of method */}
  description: {/* explanation and definition of the metaknowledge */}
  operation: {/* operation knowledge */}
  action:{ /* a course of actions */}
  rule-set:{ /* rules of the meta-knowledge */}
  ……}

4.3. Operations and properties analysis

The analysis of Petri net properties such as liveness, boundness, and reversibility is crucial for Petri net-based modeling. Fundamental techniques for such an analysis are state space construction, matrix-equation approach, and reduction or decomposition techniques. The construction of a state-space representation called a reachability graph by enumeration allows computation of all properties; however its usefulness is limited by the state-space explosion often occurring even in seemingly simple models. In spite of this capability, feasibility and efficiency considerations motivate the use of the algebraic approach and of reduction or decomposition techniques whenever possible.

In the extended, refined, or decomposed net, the properties analysis is more complicated than its original one. An alternative way is to find out whether the properties of the original net are inherited. The method of applying reduction rules to analyze a large system which reduces it to a smaller and simplified system or replaces the transitions in the net with corresponding subnets is helpful to examine the properties of the system when the properties of the smaller system are known. On the other hand, it is desirable to use an abstract model for representing a large system. Therefore, techniques are developed to transform an abstract model into a more refined model in a hierarchical way. Decomposition and refinement techniques can be used together as a top-down strategy. The former divides a large net into smaller and understandable nets, whereas the latter adds details to the existing net by replacing a component of the net with a subnet. The refinement of Petri nets may be done by replacing a place or transition with a subnet. Details about the general procedures for the refinement operations in Petri nets and the replacement rules of a place with a subnet can be found in Valette (1979); Lee and Favrel (1985); Berthelot (1986); Zhou and DiCesare (1989); and Zha (1999).

5. DESCRIPTION OF HYBRID DESIGN OBJECT MODEL

As described before, an intuitive explanation has been given on the necessity of a hybrid design object model and the necessary concepts of the knowledge representation in mechanical system and assembly design. With respect to object orientation, any modeling system can be viewed as a collection of methods used to describe the behaviors of corresponding objects. The system, in fact, is composed of a collection of objects plus methods. Design object can be generally viewed from two perspectives, namely, functionality and physical structures. However, the complete model will be represented in this section using a four-level hierarchy: function-behavior, structure, geometry, and feature.

5.1. Top-down and bottom-up design

The overall conceptual design process consists of two main stages, analysis and synthesis, which are actually top-down functional decomposition stage and bottom-up configuration stage. The top-down analysis process primarily deals with how to organize design tasks, decompose and allocate required functions, and find design solutions (e.g., suitable physical structures) for each allocated and decomposed design task and subfunctions. The bottom-up synthesis process is to synthesize and connect physical structures in terms of the relationships set forth by the corresponding functional object to compose a workable physical configuration, which is either a component or a subassembly, or the final whole assembly. At this stage, design object at last reaches its full form, design output.

A typical design of plate fastening assembly can be used as an illustration as shown in Figures 3 and 4. From a designer's point of view, an assembly is a structural model from general to detail that reflects certain relations at different levels. At the beginning of design, designers do not care about the particular structure, or no details are provided. According to the functional requirements of fastening, the units and their coupling relationships are roughly represented by a graph-like model as shown in Figure 3a. It

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

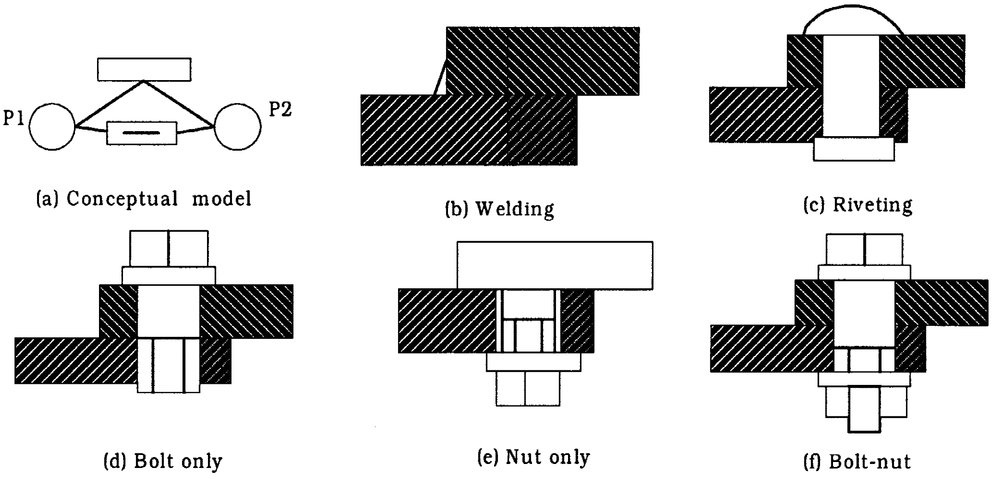


Fig. 3. Plate assembly design and its structural model with several design solutions.

is enough to represent the fastening relationship between P1 and P2 at the top level, at most with some estimated dimensions, such as the approximate diameter. Designers usually face several conceptual models to materialize the fastening structure: welding, riveting bolt-nut, bolt only, nut only, and so on, as shown in Figure 3b–e. The designer may divide the required structure thinking blocks into two sets: a bolt set and a nut set; in between is a screw fit if the

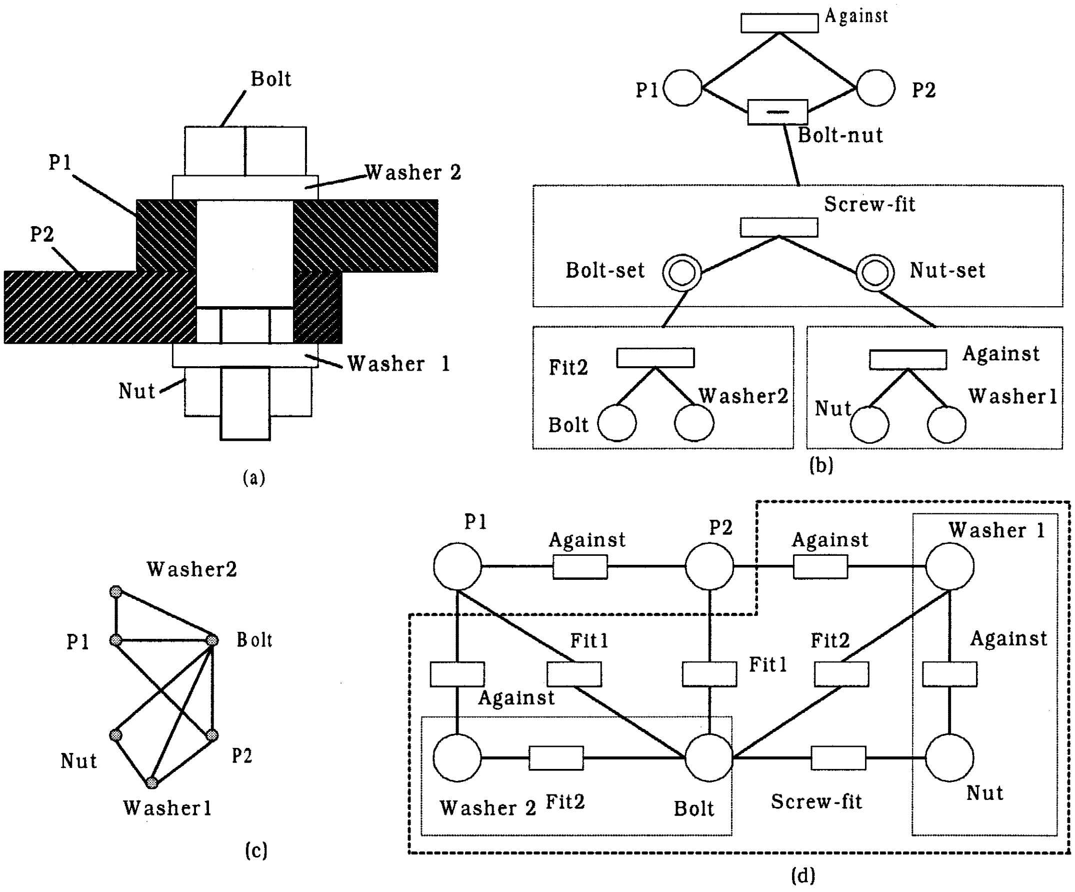


Fig. 4. A network model for top-down assembly modeling.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

designer makes a choice of bolt-nut as an initial solution, as shown in Figure 4a, b. Since the bolt-nut fasteners are quite different in different work circumstances, the devices for locking must be carefully considered in the detailed design stage. Figure 4d gives a network model based on the commonly used concepts discussed above. Thus the network model for the bolt-nut structure in Figure 4d might be recursively divided into several subnets within the dashed lines, which reflect the designer's structural thinking.

5.2. Hierarchical structure model

A mechanical system is composed of parts which are assembled to carry out specific functions. From the modular viewpoint, a key idea in design is the concept of module, called modular design. This means that there are various modules, of which interface-modules determine much of the functionality in engineering systems. For the physical structural aspect of a design object, it could be a system with many subsystems; an assembly composed of parts or components and connectors (joints); or a single part composed of physical features. Different levels of physical objects actually form a hierarchy which utilizes the relationships between different parts of the physical aspect of a design object.

5.2.1. Structural representation

Incorporating Liu's model (component-joint model) and Gui's model (component-connector model) into a hybrid one, a "place-transition" model is proposed in this paper to represent the structures of mechanical systems and assemblies, in which each part is represented as a place of P/T net and each joint is represented as a transition of P/T net. Therefore, a mechanical system (assembly) is a hierarchical P/T net, called Assembly_Model, and a subsystem (subassembly) is a sub P/T net. Using modular representation, a sub P/T net (object) can be described as either a macro place or a macro transition. This is mainly dependent on its function as either a component or a joint or a connector. Token data abstraction and dynamic distribution can be used for knowledge representation in describing the structure and system state changes.

Using object-oriented representation, the attributes and functions of the *Assembly_Model* are described as follows. The *Assembly_Model* class carries the **is-part-of** relationship of a mechanical system and its components. The attributes and methods (functions) of the *Assembly_Model* are defined to help the designer construct the structure of the mechanical system. The editing functions allow the designer to create the specific system configurations. When a designer creates a new system, the system configuration or decomposition is based on some special purposes from the designer's viewpoint. Furthermore, based on different application considerations, the designer can edit the configuration or evolve it through experiment to conceive, as a meaningful unit, an analysis that evaluates some aspects of the performance of the system or subsystem.

```
Class Assembly_Model
{
Attributes:
        ID
        Name
        Set of Assembly_Models (Nil or Composite IDs)
        Set of places (Parts) (Nil or Part IDs)
        Set of transitions (Joints) (Nil or Joint IDs)
Methods:
        Create Assembly_Model
        Add place (Part)
        Erase Place (Part)
        Add transition (Joint)
        Erase transition (Joint)
        Add Assembly_Model
        Erase Assembly_Model
        Display place (Part)
}
```

Formally, an assembly place-transition model can be defined as: $S - PTN = \{P, T, A, W\}$, where $P = (p_1, p_2, \ldots, p_m)$ is a place set which represents objects consisting of components; $T = (t_1, t_2, \ldots, t_n)$ is a transition set which represents joints; and A is an arcs set which links between components and joints; W is a weights set of the arcs. Various relations between place nodes and transition nodes in a hierarchical P/T net can be clarified with reference to Fig. 5.

1. A structure on the top level is only a P/T graph S_0 with one place or macro place node;
2. A structure on ith level $(i = 1, \ldots, L)$ S_i is a graph: $S_i = \{P_i, T_i, A_i, W_i\}$, where P_i is a set of places denoting components c_{ij} or subassemblies sub_{ik}, that is, $P_i = \{c_{ij},\ sub_{ik}\}, (j = 1, \ldots, h_i,\ k = 1, \ldots, x_i)$; T_i is a set of transitions, either joints J_{is} or connectors l_{it}, that is, $T_i = \{J_{is}, l_{it}\}$ $(s = 1, \ldots, m_i,\ t = 1, \ldots, n_i)$; A_i is a set of arcs linking J_{is} or l_{it} and c_{ij} or sub_{ik}, that is, $A_i = \{a_{isj}, a_{isk}, a_{itj}, a_{itk}\}$; W_i is a set of weights of arcs, that is, $W_i = \{w_{isj}, w_{isk}, w_{itj}, w_{itk}\}$. The structure can be also expressed by a collection of unconnected graphs $s_{i,a}$ $(a = 1, \ldots, y)$, that is, $S_i = \{s_{i,a}\} = \{\{p_i, t_i, a_i, w_i\}_a\}$, $p_i \in P_i$, $t_i \in T_i$, $a_i \in A_i$, and $w_i \in W_i$.
3. A place or transition $(p_i$ or $t_i)$ or graph $s_{i,a}$ may be associated with another graph $s_{i+1,b} = \{\{p_{i+1}, t_{i+1}, a_{i+1}, w_{i+1}\}_b\}$ $p_i \in P_i$, $t_i \in T_i$, $a_i \in A_i$, and $w_i \in W_i$ $(i = 1, \ldots, L)$. Such a graph is termed a subgraph of the graph $s_{i,a}$; vice versa, $s_{i,a}$ is termed the super graph of the graph $s_{i+1,b}$. The place or transition is termed macro place or macro transition, either a subassembly place or a connector transition.
4. Suppose an assembly P/T net that has n transitions and m places. Its assembly incidence matrix is defined as $C = [C_{ij}]$ $(1 \le i \le n,\ 1 \le j \le m)$. Every row

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

of C represents a transition or a macro transition; every column represents a place or a macro place, $C_{ij} = w\,(t_i, p_j) - w(p_j, t_i)$. For example, the assembly incidence matrix of the bolt-nut fastening assembly is as follows:

$$\mathcal{AIM}(C) = \begin{array}{r|cccccc|}
 & \mathbf{P_1(p_1)} & \mathbf{P_2(p_2)} & \mathbf{washer1(p_3)} & \mathbf{washer2(p_4)} & \mathbf{bolt(p_5)} & \mathbf{nut(p_6)} \\
\mathbf{against1(t_1)} & 1.0 & 1.0 & 0 & 0 & 0 & 0 \\
\mathbf{against2(t_2)} & 0 & 1.0 & 1.0 & 0 & 0 & 0 \\
\mathbf{against3(t_3)} & 1.0 & 0 & 1.0 & 0 & 0 & 0 \\
\mathbf{against4(t_4)} & 0 & 0 & 1.0 & 0 & 0 & 1.0 \\
\mathbf{fit1-1(t_5)} & 1.0 & 0 & 0 & 0 & 1.0 & 0 \\
\mathbf{fit1-2(t_6)} & 0 & 1.0 & 0 & 0 & 1.0 & 0 \\
\mathbf{fit2-1(t_7)} & 0 & 0 & 1.0 & 0 & 1.0 & 0 \\
\mathbf{fit2-2(t_8)} & 0 & 0 & 0 & 1.0 & 1.0 & 0 \\
\mathbf{screw-fit(t_9)} & 0 & 0 & 0 & 0 & 1.0 & 1.0
\end{array}$$

Thus, any machine can be viewed as consisting of two basic classes of objects: a class of places (components) and a class of transitions (joints). P/T net can model mechanical causal relations between components. The main purpose of transitions is to make places (components) work normally through connecting these components. For example, a transition with a motion transmission function might become a gear pair; a transition with a fixing function might be a collection of geometric mating surfaces such as a cylinder and shoulder. Since places and transitions for components and connectors are conceptually fuzzy, they might form a fuzzy P/T net which represents a subassembly during later stages of design.

The hierarchical model as suggested has the advantage of "implicitly incorporating abstraction and refinement." A multi-level P/T net could be imagined to be generated by network modeling from top to bottom. To implement a number of levels of abstraction, a usual network model in a macro place or transition is split into several embedded blocks corresponding to the designer's thinking patterns or thinking blocks on various levels.

For the example in Figure 4a,"bolt-nut" structure, the designer's goal of fixing plates P1 and P2 is expressed as putting a connector between the component P1 and P2 on the top level. Thus a macro transition (labeled rectangular) located between P1 and P2 in Figure 6 replaces the thinking block on the first level after the first abstraction. A further splitting with the second thinking block might be "bolt-set" and "nut-set" which are logically components. Two places

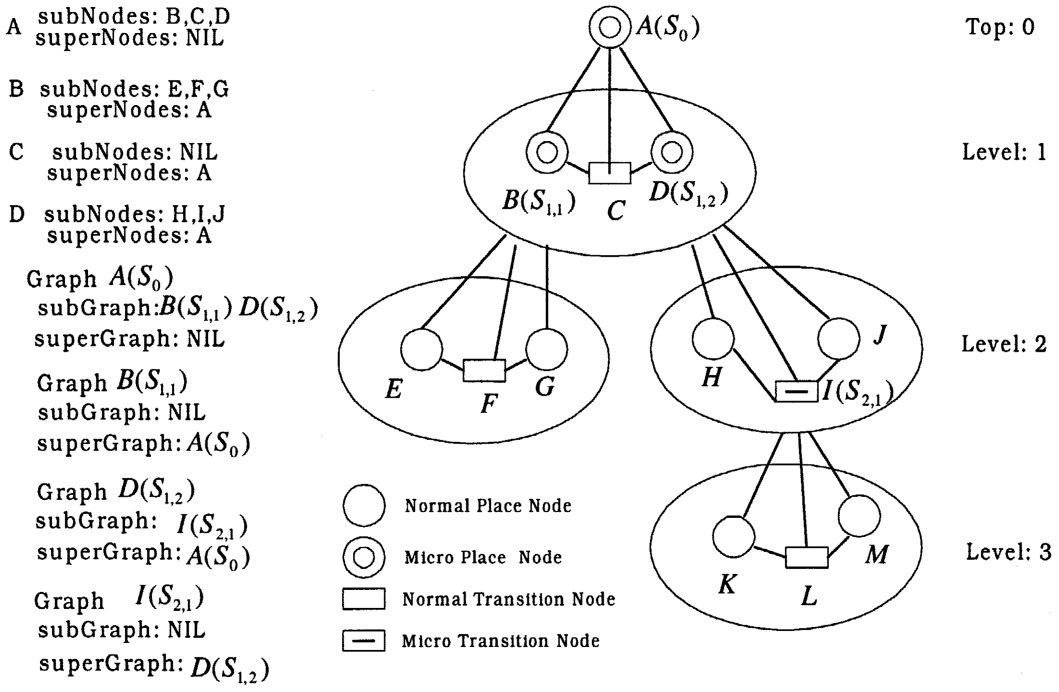


Fig. 5. Hierarchical P/T model.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

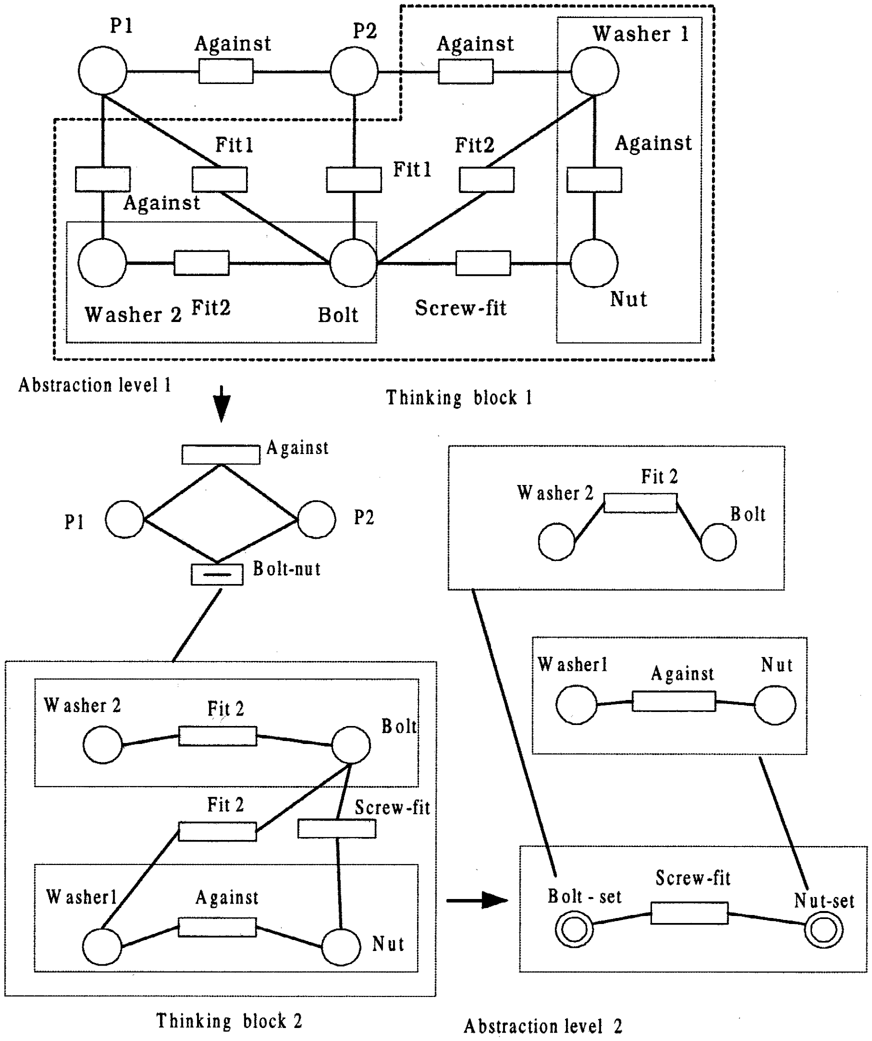


Fig. 6. A multi-level P/T net graph for the "bolt-nut" structure.

(labeled circles) replace the thinking block on the second level after the second abstraction. Replacement of a part of the original network model by a logic node is a type of abstraction through which nodes in the network model are pulled down. In this way, a multilevel Petri net graph with increasing structural information detail could be naturally created during in-progress design. The mechanism to generate a multi-level P/T net graph has been realized in the experimental prototype assembly modeling system which will be described in Section 7.

The significant improvements created by using a multi-level P/T net lie in two aspects: Nodes of a multilevel P/T net graph are functionally divided into components and joints only and graphically distinguished into normal nodes which denote atomic components and macro nodes which are associated with another level P/T net graph. Such a distinction reflects various levels of abstractions at different stages of a design process. One of the advantages of using a high abstraction of connectors, for example, is that feature-based modeling for single-piece parts becomes a natural extension of assembly modeling. On the lowest level, connectors are those features of single components that mate and structure these components.

On the other hand, places, transitions, and P/T nets not only can perform functions in possible behaviors, but also can show behaviors in possible properties as well. Behaviors (states) are time-varying properties. Usual invariable properties of places or transitions are typically performance, form, size, color, stability, manufacturability and assemblability, transportability, suitability for storage, and so forth. They can be represented in the form of linguistic variables and allow for the possibility that the attributes take values in a base range. Therefore, an incomplete structure based on places and transitions for components and connectors in the early design stage can be modeled through imprecise descriptions of their properties. During design process, places and transitions are refined until their properties are all certainly defined.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

5.2.2. *Abstraction and task-oriented schema*

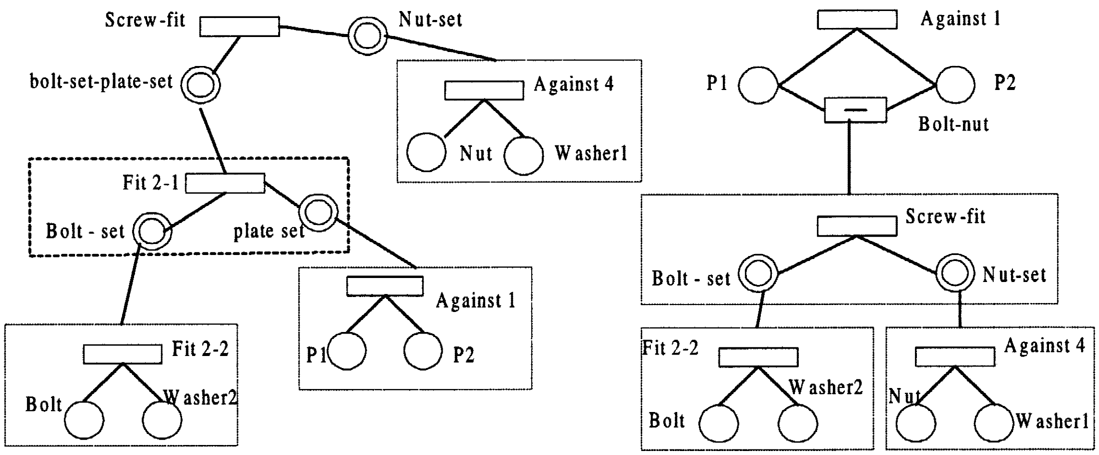
One of the main objectives of the proposed hierarchical P/T net structure model is to support the user to work at the abstract level. The abstraction is a way to specify the functional aspects of a system and provide a mechanism for ignoring irrelevant data or detail by use of a specific analysis procedure. However, the most meaningful way to configure mechanical systems and assemblies is in a form that allows a designer to conceive some analysis that could evaluate aspects of the performance of the system or subsystem as a whole. Depending on the purpose, the model can be examined in different ways: The designer may want to know the kinematic relationships between different parts and kinematic configuration for the influence on kinematic performance (e.g., modular robot); the manufacturing engineer may want to examine the configuration with emphasis on the assembly process (e.g., assembly sequence) and assembly system.

Both hierarchical structures in Figure 7 represent the same mechanical system or product, but they emphasize different viewpoints. This feature helps the user to work at different levels of conceptual abstraction. Also, the data model can support the user to work at abstract geometric views in a top-down design environment, despite the details contained in the parts and joints yet to be introduced. Regardless of the actual geometry of the parts, a mechanism can be described functionally by the kinematic pairs and connections between them. By assigning proper dynamic properties (methods) to the *Assembly_ Model* (e.g., create, modify, delete, and add objects), the user can easily create a distinct configuration of the system structure, and even edit the structure, throughout the design stages.

5.3. Functions and behaviors in design object model

5.3.1. *Functional and behavioral description*

Taking the structural model as a framework, the functional and behavioral descriptions can be further constructed for a mechanical system or machine. Function, as usual, means what the system is for. It represents input/ output relations of the system to the outside. Functions of a machine can be viewed as deriving mainly from user requirements, independent of any particular solution. Behavior (state) means how the system works. When any machine performs human-desired functions, it operates with changes in its location, shape, attributes, or the relations between its parts. The change of its states represents its behavior. Actually, almost all mechanical systems are based on physical processes which are in turn based on physical effects. These effects can be formulated by means of the physical laws governing the physical quantities involved. The term "design intent" or "purpose," which is conceptually similar to "function" and "behavior," is widely used. Different understanding of what these similar terms mean will influence the functional and behavioral modeling. The distinction between "behavior" and "function" is favorable to clarify for design problem-solving, that is, a functional model for a required machine is mainly related to a design task structure while a behavioral model on the basis of physical laws is related to a particular solution. Functions of the required machine are not the subset of the machine behaviors. In the design decision-making process, designers are mostly interested in their qualitative behaviors. The design process can be viewed as transforming a functional model to a design solution through its behavioral modeling.



(a) Hierarchical bolt-nut fastening P/T net (assembly process oriented)

(b) Hierarchical bolt-nut fastening P/T net (design oriented)

Fig. 7. Different system configurations of bolt-nut fastening.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

A formal approach to representing a design functionally is given in the form of a function block diagram, as shown in Figure 8. The term function logic applies to the reasoning used to develop and use the function block diagram. Originally developed to stimulate design creativity, this systematic approach to the representation of design process at a high level relies on identification of design goals and describes them in functional terms. The resulting, compact description is called the basic function of the design. The basic function is decomposed by the design team into several functions, which collectively perform the function. These secondary functions are then translated into components or recursively decomposed. The function decomposition process continues until one can map each function into a component or system that will accomplish it. Such results are for the most part preliminary because practical designs rarely feature a one-to-one correspondence between functions and components.

5.3.2. *Functional representation*

Function representation refers to the mapping of a function into data structures in a computer program, plus the representation of procedures, methods, and mechanisms used to access, manipulate, and utilize that function. The four-step procedure for function representation is elaborated as from understanding, abstracting, and describing to representing the function concisely and rigorously. As a design intention or an inherent attribute of a product, function normally cannot be represented numerically, except for some specific situations, where a function is involved in processing of energy flow, material, or signal, or some physical quantities related to energy, and the relationship between input and output of flow is quantitatively or mathematically known to the designers. For example, if a function is "to reduce the angular speed to 1:3," it can be mathematically described as $3w_{out} = w_{in}$. This kind of function is sometimes regarded as "the transformation from input to output."

The most commonly used method for function representation is describing a function using a natural language approach, in which a function is a predicate which specifies a relationship between input and output from the physical structure: $\mathbf{apply}(\{d_i\}, S_i) \rightarrow P_1 \wedge P_2 \ldots \wedge P_n$, where S_i is the structure under consideration; $\{d_i\}$ is a set of input parameters applied to the structure; predicate **apply** relates to these parameters and P_i $(i = 1,2,\ldots,n)$ are predicates by tasks. From the function logic in hierarchy, the function and function decomposition actions can be represented with predicates in Prolog (Zha, 1999). While this is the actual language that is used to derive the functions, there also exists a method to graphically model functions. Based on the relationship between proposition logic and Petri nets, these models are referred to as Petri nets, and can sometimes give a more clear and intuitive representation for functions. Note that using a Petri net graph is not intended to replace the

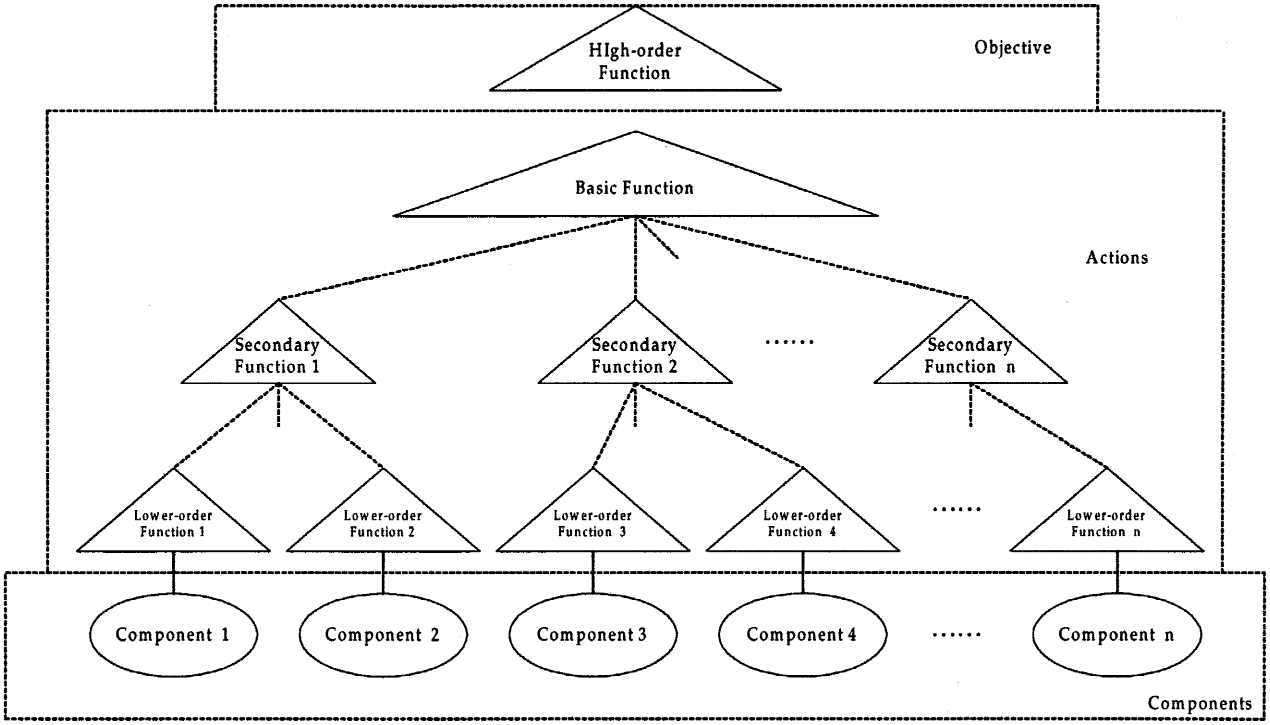


Fig. 8. Hierarchical structure of functional aspect of a design object (function logic diagram).

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

prolog code, but rather provides a different representation of functions and their mappings.

A function logic model represented by a place-transition Petri net can be defined as : $F - PTPN = \{P, T, A, W\}$, where $P = (p_1, p_2, \ldots, p_m)$ is a place set which represents functional objects consisting of hierarchical subfunction components; $T = (t_1, t_2, \ldots, t_n)$, is a transition set which represents function decomposition actions; A is an arcs set which links between functions and their decomposition actions; and W is a weights set of the arcs. Figure 9 gives an example of functional Petri nets model and prolog codes.

To avoid ambiguity, knowledge relating the function to its working scenario should be added, and several relevant issues have to be represented together with this representation (Deng et al., 1998). For this purpose, an object-oriented knowledge Petri net representation scheme can be used for function representation. The most generic function can be represented as the top-most functional object. The class of these objects is as follows:

```
Class Function (macro plate or transition)
{
Name: Transition/place
  Complement: Additional information or knowledge annotation
  Type: Performance/Assembly/Manufacturing/
        Maintenance/Others
  Level: Overall/Embodiment/Geometric/Feature
}
```

The name attribute is expressed by the transition and place variables, where transition is used to describe the action relating to the function, for example, "transmit," "change," and so on, which is normally a verb; place is used to describe the target of the action. For example, if the function object name is "to reduce vibration," then the target of action "reduce" is "vibration." Complement attribute is used to offer some additional information or knowledge annotation to the name attribute when necessary.

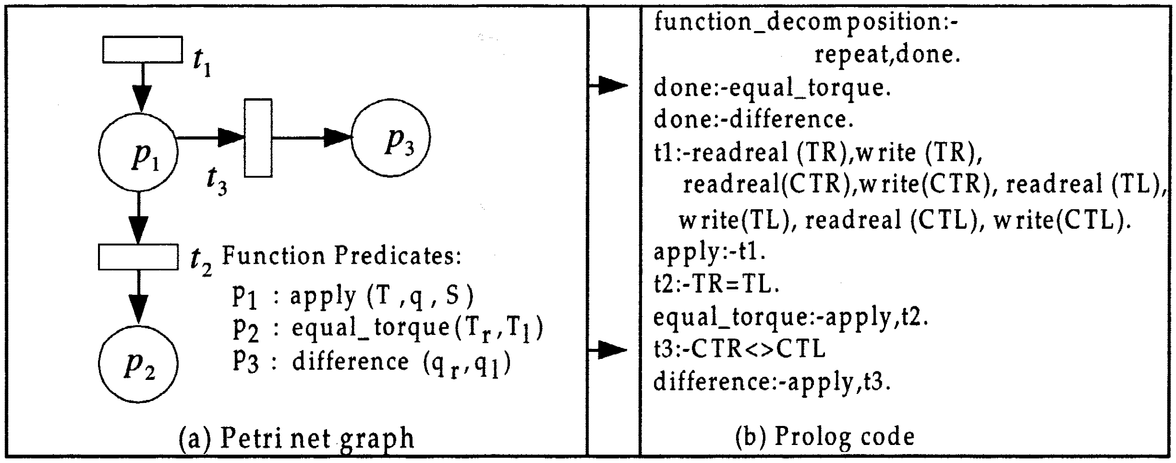
With this top-level function representation, specific function can be represented as its child object, which can inherit its attributes, but add some specific attributes pertinent to the child object. For example, if the specific function happens to be a fundamental mechanical function, then this function object can be represented as:

```
Class Fundamental_Mechanical_Function (macro place)
{
inherit: Function
    category: {Supplying/Storing}
              {Transmitting_Motion/Force/Material}
              {Converging/Branching}
              {Changing_Form/Magnitude}
}
```

Furthermore, if the specific function is "to slow down the angular speed by 1/3," which is also a fundamental mechanical function, then this function can be represented as:

```
Class Specific_Function_1 (place)
{
  inherit: Fundamental_Mechanical_Function
  variable: Input_angular_speed
  variable: Speed_ratio
  method: derive_Output_angular_speed
        { variable: Output_angular_speed
          Output_angular_speed = Input_angular_speed *
          Speed_ratio
          Output: output_angular_speed
}. . .   . . .}
```

This function Specific_Function_1 inherits the attributes from its super or parent function, Fundamental_Mechanical_Function, while at the same time, Fundamental_Mechanical_Function also inherits attributes from the top-



t_1
p_1
t_3
p_3
t_2
p_2
Function Predicates:
P_1 : apply (T , q , S)
P_2 : equal_torque(T_r,T_l)
P_3 : difference (q_r,q_l)
(a) Petri net graph
(b) Prolog code

```
function_decomposition:-
                repeat,done.
done:-equal_torque.
done:-difference.
t1:-readreal (TR),write (TR),
   readreal(CTR),write(CTR), readreal (TL),
   write(TL), readreal (CTL), write(CTL).
apply:-t1.
t2:-TR=TL.
equal_torque:-apply,t2.
t3:-CTR<>CTL
difference:-apply,t3.
```

Fig. 9. Functional Petri nets model and prolog codes.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

most function. This makes the function Specific_Function_1 automatically have all the attributes from its upper functions; a method is encapsulated in this function to characterize the relationship between two variables Input_angular_speed and Output_angular_speed. This feature from object-oriented technology makes it far superior to the traditional frame-based approach, where only attribute data can be grouped but cannot be encapsulated. The relationship between different level of function classes and the relationships between function classes and function objects and instances of the class can be illustrated as shown in Figure 10.

Because every fundamental mechanical function has its corresponding physical structures or design elements that can perform it, this data or information can be represented inside its function class. For simplicity, if several functions, either coupled or not coupled, are performed by one design element, then they can be represented in one function class. Thus the class for fundamental mechanical function presented above can be extended as follows:

```
Class Fundamental_Mechanical_Function (macro place)
{
inherit: Function
    category: {Supplying/Storing}
              {Transmitting_Motion/Force/Material}
              {Converging/Branching}
              {Changing_Form/Magnitude}
    structure: Design_element
}
```

5.3.3. Behavioral modeling

As a functional behavior reflects the states of the structure, the behavioral modeling is simultaneously related to function representation. The behavioral modeling process synthesizes structure and functions from a state-based model through qualitative representation based on machine-centered ontology and/or process-centered ontology (Gui, 1993). The designer can make use of function knowledge representations to incrementally describe a behavioral specification of a mechanical system. This behavior will be later interpreted as a knowledge-based Petri net model and then analyzed for consistency checking and function or property verification.

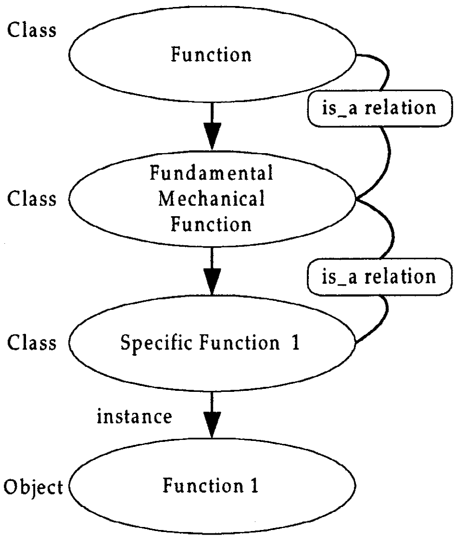


Fig. 10. Hierarchy of function class and relationship between class and object.

In qualitative modeling, parameters can only take on one of a small number of values. This set of possible values is determined by the quantity space which is represented by an ordered set of discrete landmarks, for example, $(-\ 0\ +)$ or $(-\infty\ 0\ \infty)$. The representation for a parameter is a quantity associated with two numbers: a qualitative magnitude and its derivative (e.g., increasing, decreasing, or constant). Because there are fewer constraints to the parameter values in qualitative modeling, there inevitably exists ambiguity. The behavioral modeling problem can be described as follows.

Suppose that $S_t = \{s_1, \ldots, s_n\}$ is a collection of state diagrams, where each state diagram s_i is a bi-tuple $[\{s_{ij}\}, \{t_{ij}\}, \{e_{ij}\}]$. $\{s_{ij}\}$ is a set of possible qualitative states presented as a tuple of qualitative values for the structure parameters and their variations. t_{ij} is a set of state transitions: $t_i \subset s_i \times s_i$. $\{e_{ij}\}$ is a set of events for state transitions. A behavior is a subset of B: $\mathrm{b} = \{(s_i)$— $\forall \mathrm{i}\ s_i \in S_t\} \in \mathrm{B}$. The acceptable behaviors can only contain acceptable states and state transitions: $B_{accept} \subset \{\mathrm{b} = (s_i) \in \mathrm{B}$—$\forall \mathrm{i}\ s_i \in S_{t,accept} \wedge t_i \in T_{accept}\}$.

There are some efficient behavioral modeling techniques developed based on the state-based models such as bond graph theory of system dynamics (Ermer et al., 1993; Gui, 1993) and finite state machine. These methods can combine the advantages of process-centered ontology with the modeling features of the machine-centered approach. However, they cannot explicitly express the notions of concurrency, causality, and conflict. A Petri net can naturally capture these notations. Recent research indicates that the the Petri net method has great potential for qualitative and quantity reasoning in discrete, continuous, and hybrid (discrete and continuous) systems and the preliminary design of machines (Valette et al., 1994). A Petri net approach to behavioral modeling is proposed in this paper.

The proposed approach first applies state-based models to model the machine behaviors and then applies algorithms 1 and 2 of synthesizing Petri nets from state-based models to transform the transition system (TS) models to behavioral Petri net models (Cortadella et al., 1995, Zha, 1999). In the course of transformation, the underlying features of knowledge Petri net models are incorporated into the place-transition structural model. Suppose that the behaviors of a machine are described as states, S, and state transitions, T, under events, E, that is, $B = (S, E, T)$. The

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

corresponding behavior transition system can be described as $BTS = (S, E, T, s_{in})$, where S is a finite nonempty set of states, E is a set of events, $T \subset S \times E \times S$ is a transition relation, and s_{in} is an initial state. The elements of T are often denoted by $s \xrightarrow{e} s'$ instead of (s, e, s'). The reachability relation between states is the transitive closure of the transitions T. If there is a (possibly empty) sequence of transition s between states s and s', then this is denoted by $s \xrightarrow{\sigma} s'$ or simply by $s \xrightarrow{*} s'$. A behavior TS is called deterministic if for each state s and each label a there can be at most one state s' such that $s \xrightarrow{a} s'$. Otherwise, it is called nondeterministic. The BTS can therefore be represented as an arc-labeled directed graph and Petri nets. Figure 11 gives a simple example to demonstrate the procedures of behavior modeling.

Algorithm 1: Mapping from a elementary behavior transition system to a Petri net

Input: a TS

Output: a PN

```
Procedure: map_to_PN
{
Step 1: For each event $a$ a transition labeled with $a$ is generated in the PN;
Step 2: For each (minimal) region $r_i$ a place $p_i$ is generated;
Step 3: Place $p_i$ contains a token in the initial marking $m_0$ if
        the corresponding region $r_i$ contains the initial state of
        the EBTS $s_{in}$;
Step 4: The flow relation is as follows: a $\in p_i\bullet$ (output transition of $p_i$) if $r_i$ is a pre-region of $a$ and $a \in \bullet p_i$ (input
        transition of $p_i$) if $r_i$ is a post-region of $a$.
Step 5: end}
```

Algorithm 2: Synthesizing behavior Petri net from state-based models

Input: a behavior TS

Output: a behavior PN

```
Procedure {
begin
    repeat /* Generation of pre-regions and label splitting */
    split := false;
    for each $e \in E$ do
        $e^o$ = expand_states (GER(e),0); /* GER is generalized excitation region */
          if $\neg$ excitation_closure ($e^o$) then
              split_labels (e);
              split := true;
          end if
    end for
    until $\neg$ split;
      find _irredundant _cover;
      map_to_PN;
end }
expand_states (r,R)
{
begin
      /* r is the set of states to be expanded */
      /* R collects all regions generated */
      if r is a region then
        R = R $\cup$ {r};
        Return;
        /* since any region expanded from r would not be minimal */
      end if;
      find $e \in E$ violating some region condition in $r$;
```

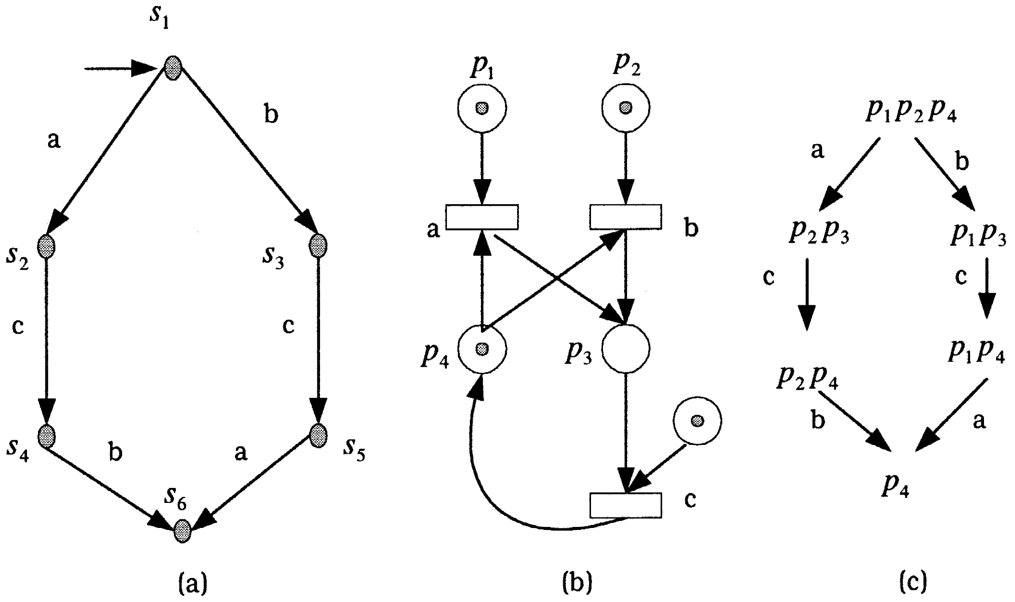


Fig. 11. An example of transition system (a), the corresponding Petri net (b), and its labeled reachability graph RG (c).

```
    $r' = r \cup$ {1st set of states to legalize e};
    expand_states ($r'$,R);
    /* for some conditions the set of states must be expanded in two directions */
    $r' = r \cup$ {2nd set of states to legalize e };
    expand_states ($r'$,R);
end }
```

Based on the discussion above, modeling behaviors of a complex physical system by knowledge Petri nets only requires a small number of components and rules that various computer programs can easily implement. Qualitative descriptions of functions and behaviors (i.e., static states) from the device-ontology point of view can be embedded in the place-transition assembly structural model as predicate logic or more generally as fuzzy logic when reasoning with incomplete structure. The representation of input-output causality (i.e., state change or transitions) from the process ontology point of view can be established and explained using Petri nets from different perspectives and modeling levels. It is more concise and clearer than causal networks which are currently used in most of the proposals. More importantly, the mapping from assembly structural P/T net to behavioral Petri net can be easily implemented.

5.3.4. Elemental functions and P/T units

Although no function in structure nor structure in function exists universally, fundamental or elemental mechanical functions with typical connector forms are widely used in mechanical structure. The main idea underlying a design prototype is to represent a class of generalized heterogeneous groups of elements derived from similar design cases that provide the basis for the start and continuation of a design.

Similar design cases can be viewed as design prototypes on the most basic level of mechanical design. From a functional point of view, connectors provide actions in the sense that two jointed components can perform the required functions. Some basic functions such as fix, motion constraint and motion/force transmission, material transmission, and power supplier are listed in Table 1. Elemental mechanical functions and behaviors, which are suitable for any mechanical systems, have their own expressions in predicates, and their possible unit forms in P/T nets.

Different forms with the same function may show dissimilar behaviors. Fixing with a nut and bolt works by transmitting the axis load to the fastened parts; while fixing with a key fit is based on transmitting the shared load to the interference parts. They can be distinguished in the

Table 1. *Elemental functions, behaviors, connector forms and P/T units*

Function	Behavior	Connector form	P/T unit form
Fix	Fastening	Fastener(bolt_and_nut, bolt_only, nut_only,rivet, welding, solder, etc)	(Macro)places, or(macro)transitions
	Coupling	Key_fit,pin_fit,screw_fit	Transitions
	Latching	Snap_ring	Place
	Attaching	Fix_fit1,Fix_fit2, Fix_fit4, against	Transitions
Motion constraint	Rolling	Journal_bearing, Roll_fit1,Roll_fit2,	(Macro)places, or(macro)transitions
	Sliding	Cylinder guide keyway spline	Transition
	Screwing	Screw linkage_c_joint	Transition
	Motion converting	Cam	(Macro)places, or(macro)transitions
Motion/force transmission	Sealing	Rubber_ring circlip	(Marco) place
	Linear_limit	Spring	Place
	Reduction	Gear_mesh,screw_mesh, belt_mesh	(Macro)places, or(macro)transitions
	Clutch	Clutch_disk	Place
	Pumping	Cylinder	Place
Material transmission	Piping	Pipe	Place
	Material flow changing	Valve	(Macro)place
Power supplier	Providing mechanical power	Motor	(Macro) place

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

following form of predicates: the motion predicates for fix function Fix$(S_a,S_b) \rightarrow$ motion(S_a) = motion(S_b), and the predicates for behavior of fastening and coupling, Stress_monotonic_increase(+,Length(+l)), Relative_length(l) ≤ allowable_limitation(Max), Stress_monotonic_decrease(−, Area(−A)), respectively. In types of rotation assemblies, the states of gear meshes may be changeable. Typical examples are shifted gear units as shown in Figure 12, where the predicates for their behaviors can be modeled as Shift_mesh$(S_a,S_b,$ mesh_state), Mesh_state(+,0) or (0,+) for S_a, S_b: 2-connected gear units, Mesh-_state(+,0,0) or (0,+,0) or (0,0,+) for S_a,S_b: 3-connected gear units.

5.4. Geometries in design object model

The geometric model is a formal logical/mathematical representation of shape and size of a part. Most geometric models are presented as solid models. Since the interest is in reasoning about 3D objects, solid modeling has been chosen to create the geometric models of individual parts of the assembly. Other common representational methods, such as wire-frame systems, instances and parameterized shapes, constructive solid geometry (CSG), and boundary representation (B-rep) are also used to model mechanical parts in the geometric database of CAD systems. Recent research work has been carried out on the addition of tolerance of information, dimensional or geometric, to the part solid model.

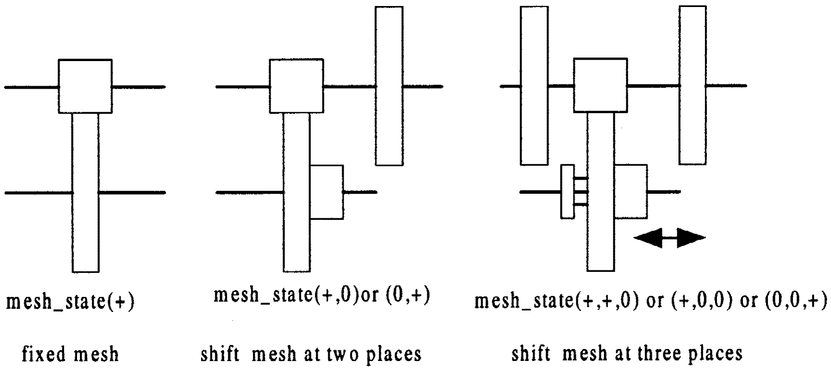
Solid modeling, features, and attribute relationships are the basis for more complete product definition. In addition to rigorously defining geometry and topology of individual parts and joints above, product assemblies are defined through solids primitive modeling by defining the following:

1. Instances or occurrences of each part in a hierarchical manner;
2. The relative location of each instance or occurrence of the part in terms of the part's *x*, *y*, and *z* coordinates relative to the assembly's base or reference point *x*, *y*, and *z* coordinates;
3. For each instance or occurrence of a part, the part's orientation in relation to the assembly's orientation;
4. Vectors or axes of rotation and translation to describe movement of parts within assemblies.

A geometric modeling or construction indeed can be seen as a series of successive operations, consisting of instantiation of geometric entities (e.g., points, lines, circles, etc.), which will be used to create new entities with simple constructive methods until the desired result is reached. The user interface enables users to interact with the solid and the entities that constitute it, enabling transformations and maintaining relationships among objects, that is, the geometric constraints.

To make explicit the process underlying a geometric construction and its successive manipulations, a kind of knowledge Petri net graph, called GC-PN (Geometric Construction Petri net graph) was defined that describes the sequence of steps that enables the designer to generate a geometric entity and the relationships among the constituent objects (Zha, 1997; Zha, 1999). The system can maintain an internal representation of the GC-PN and turns to it for constraint solving during transformations. GC-PN is helpful in describing informally and in an intuitive way a problem which has intrinsic concurrent aspects. Based on geometric construction Petri net formalisms, a geometric modeler, *GeoObj* (Zha, 1997; Zha, 1999), was developed in object-oriented C/C++ language as an extension of its predefined classes defined to represent Euclidean geometric knowledge and implement primitive geometric entities and the hierarchy among them.

This approach can yield a complete definition of the product's geometry and topology at any level in the product structure. Many assembly relationships (e.g., topological liaison, geometric liaison) and constraints (e.g., geometric constraints, and partial precedence constraints) discussed in assembly planning (Zha, 1998a) are extracted or reasoned out from the defined assembly geometric model.



mesh_state(+) fixed mesh

mesh_state(+,0)or (0,+) shift mesh at two places

mesh_state(+,+,0) or (+,0,0) or (0,0,+) shift mesh at three places

Fig. 12. Modeling behaviors of the shifted gear meshes.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

5.5. Features and semantics in design object model

In the design object model developed, the form feature, precision features, and assembly features are organized in the mechanical system's hierarchical structure. Form features and precision features are embedded in the part object, while assembly features are carried by the joint object. Form features are the geometric features which are designated to represent the part's shapes. A form feature is carried by the geometric representation of the part. Precision features include tolerances and surface texture, which are also grouped under the same composite attribute (geometric representation). Precision features are used to describe the final product design information for CAPP/CAM tools to select processes, machine tools, and tooling. Assembly features are particular form features that affect assembly operations. Each form feature has certain precision features associated with it. For example, a slot (form feature) has dimensions such as height, width, and length; each dimension has tolerance (e.g., positional tolerance, straightness, or perpendicularity to some datum) and surface finish (e.g., lay direction, average surface roughness). When parts mate together, both the parts' form features and precision features govern the assembly operations. The parts' form features directly affect the joining conditions, for instance, a hole or pin indicates the fit condition; a threaded stud or threaded hole suggest a torque operation is needed. Obviously, the precision features of the mating parts also affect the quality and manufacturing processes of the assembly. Figure 13 shows a geometric model of a model product with features.

It has been shown that feature-based product models for assembly can help considerably in both assembly modeling and planning, on the one hand by integrating single-part and assembly modeling, and on the other hand by integrating modeling and planning. For specific assembly-related information, assembly features are used in which handling features contain information for handling components, but connection features contain information on connections between components. For product and its assembly process, both part-level information including name, identification, type, class, material, heat-treatment, and geometric representation and feature-level information including name, identification, type, parameters, locations, tolerances, relations, and surface-finish are required. Three classes are defined for describing the product design as follows (Gu & Yan, 1996):

Geometric Entity: its super class-object
Sub-class-part and feature
Instance variables
Name: the unique identifier
Type: the sort of the object
Part: its super class—Geometric Entity
Sub-class-feature
Instance variables
Component: features a part holds
Neighbor: related part's names
Relating component: features having relations with other parts
n-relation: related part along a specific direction, n, and the position of the relating feature, where n = ±X, ±Y, ±Z

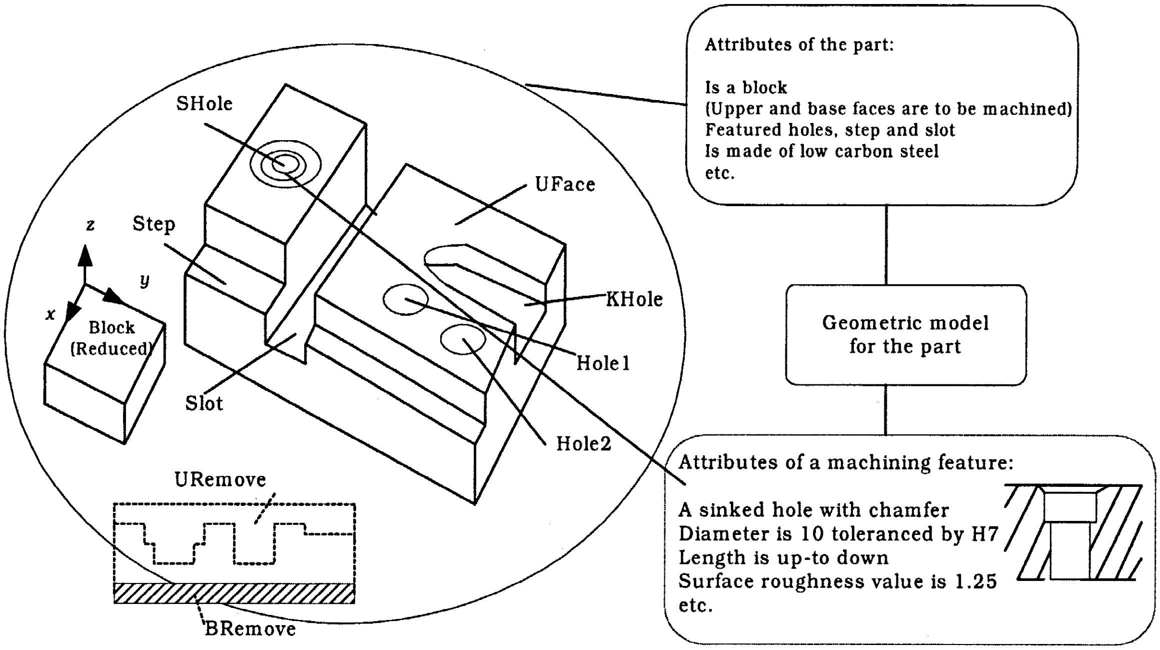

Attributes of the part:
Is a block
(Upper and base faces are to be machined)
Featured holes, step and slot
Is made of low carbon steel
etc.
SHole
UFace
z
Step
y
x
Block
(Reduced)
KHole
Hole1
Slot
Hole2
URemove
BRemove
Geometric model
for the part
Attributes of a machining feature:
A sinked hole with chamfer
Diameter is 10 toleranced by H7
Length is up-to down
Surface roughness value is 1.25
etc.

Fig. 13. Geometry and form feature of the part.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

n-list: a list of features whose normal is n with the order from nearest to farthest along n, where n = ±X,±Y,±Z

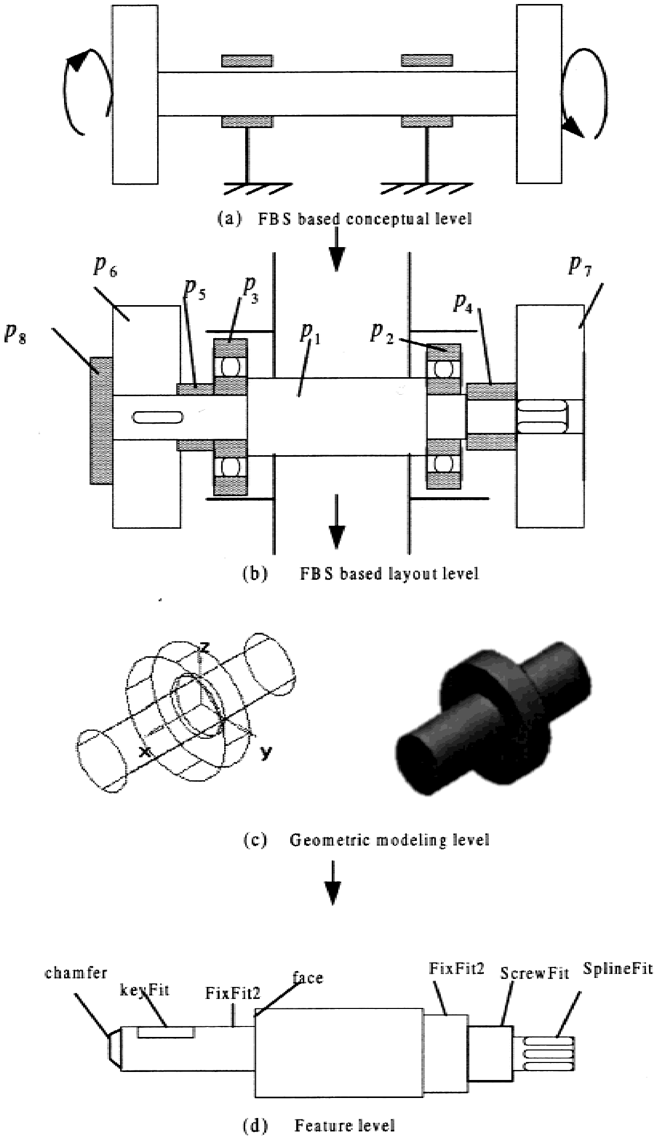
Feature: its super class—Geometric Entity

Instance variables

Location: position (x,y,z) and orientation (nX,nY,nZ) of a feature

Relation: related part name

One of the advantages of using a high abstraction of connectors or joints is that feature-based modeling for single-piece parts becomes a natural extension of assembly modeling. On the lowest level, connectors or joints can be considered as the features of single components that mate and structure components. In Figure 14, the feature model for a step shaft in an assembly can be thought of as consisting of the shaft and a set of connectors: keyFit, Spline Fit, and several cases of FixFit, all of which are features as usual. Thus a unified description of a feature-based model of both an assembly and single piece components will be obtained through this data abstraction of components and connectors on various levels from function-behavior-structure based conceptual modeling, geometric modeling to feature-based design. The feature links of an axle system using P/T net is shown in Figure 15.


(a) FBS based conceptual level

(b) FBS based layout level

(c) Geometric modeling level

(d) Feature level

Fig. 14. Data abstraction at different assembly levels.

5.6. Constraints in design object model

The design process, as represented through constraint modeling, is goal directed. The process sets out to solve problems by the resolution of conflicts. The constraint Petri net is used to model parametric and constraint design processes. The constraint network is a collection of constraints transitions that are connected by virtue of sharing variables (places). The value of a variable that is linked to a constraint may influence the values of other linked variables. In this manner changes may propagate throughout a network. This ability to propagate makes constraint networks unique and enables the network to support nondirectional inference. A user of a constraint Petri net can observe the changes made in linked variables as the initial change is made. This makes it a powerful tool in modeling the relatively ill-structured assembly design problem.

Constraint modeling using Petri net formalisms can be illustrated by shaft assembly in a gearbox. The requirement of shaft assembly is complete fit of shaft and bearing cap, as shown in Figure 16a. Therefore, the assembly constraints can be described as: $b = f$, $e > b$, $a < b$, $f > i$, $i > a$, $c > h_b + h_i + j$, $j < g + h$, $h = c - h_b - h_i$, $h_e \geq 0$, $d > 0$, and so forth. Accordingly, the partial dimension constraint graph can be shown in Figure 16(b). Once the problem becomes unordered, the Petri net becomes complex due to the necessary inclusion of the control places and interactive transitions. A solution for the above problem uses constraint modeling procedures within each of the transition activities. In the ordinary Petri net representation of the function: $C = A + B$, shown in Figure 17a, tokens are placed in A and B in order to fire the transition. The final state is given by a token in C. Based on the constraint modeling, the constraint transition contains not an equality but a constraint rule (or number of rules). Thus the above relationship is rewritten to: Rule $A + B - C$, shown in Figure 17(b), and is true when the rule equates to zero. The constraint transition is fired when two tokens are present in any of the data places and so the third can be found. The solution is carried out automatically by software solving constraints. The above two cases can be generalized and described as a macro constraint transition, as shown in Figure 17c. More details about the knowledge based Petri net constraint modeling was discussed in Zha (1999).

The development of methods for generating flexible models which can be modified dynamically is a critical problem to be solved in feature-based design. Constraint-based

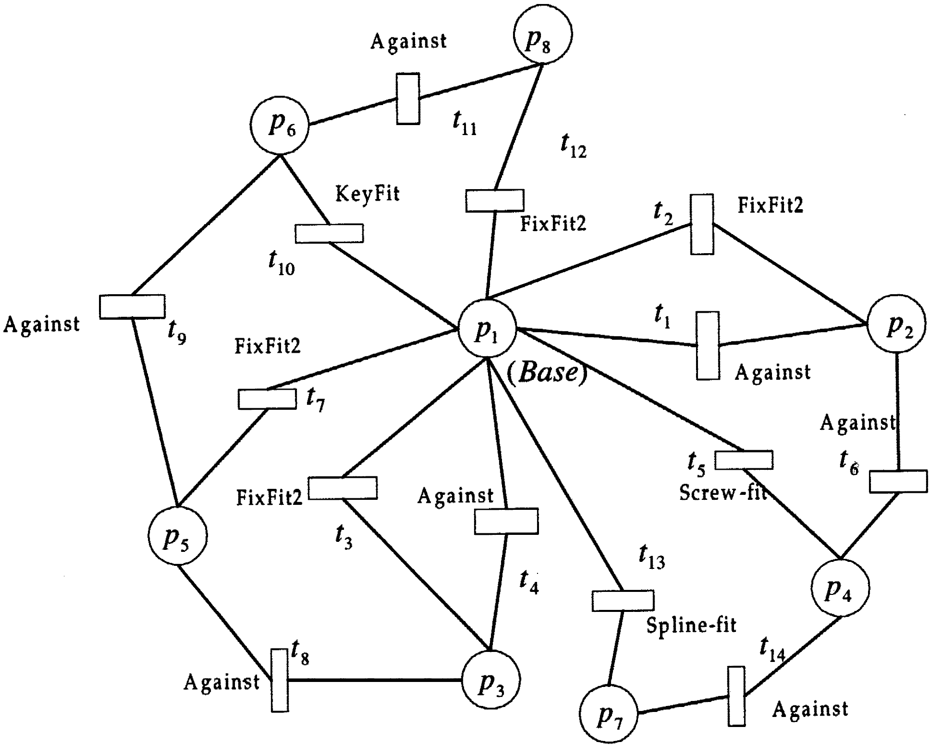


Fig. 15. Feature-links of an axle system represented by P/T net.

dimension-driven geometry through constructive schemes using Petri nets is a useful method for design modifications. The algorithm for dimension variation of feature-based models used in this paper is based on B-rep/CSG hybrid scheme and operates directly on B-rep (Zha, 1999). The dimension variation of models has very high efficiency. The production information including the features locating dimensions and other data for manufacturing will not be lost after model variation and modification. Furthermore, the definition and solution of features constraints and chain reactions of constrained features are also supported. This algorithm is an organic combination of feature repre-

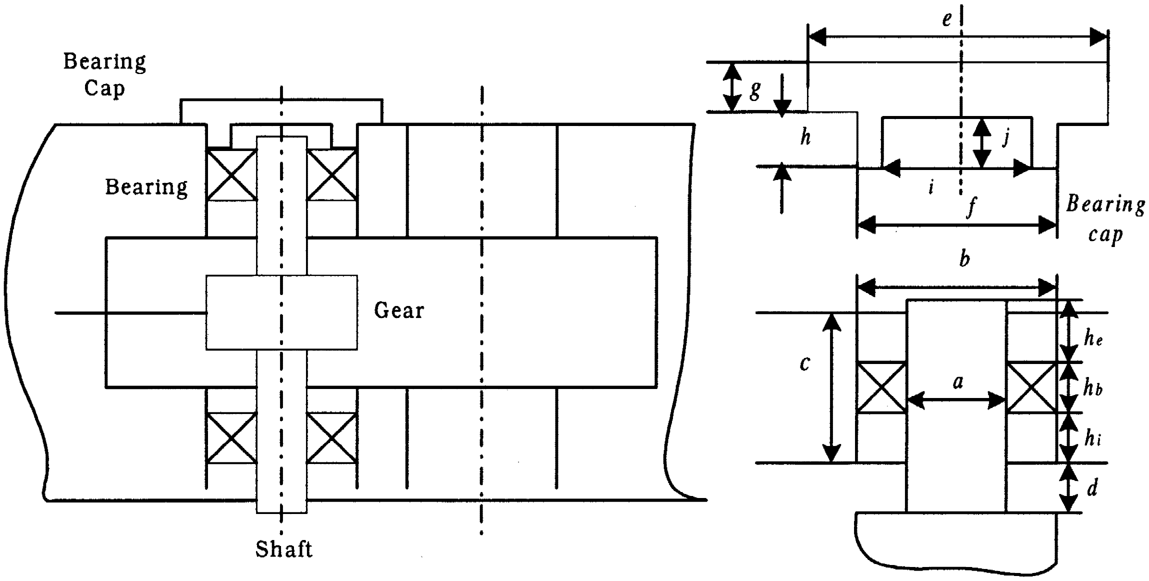


Fig. 16a. Shaft assembly.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

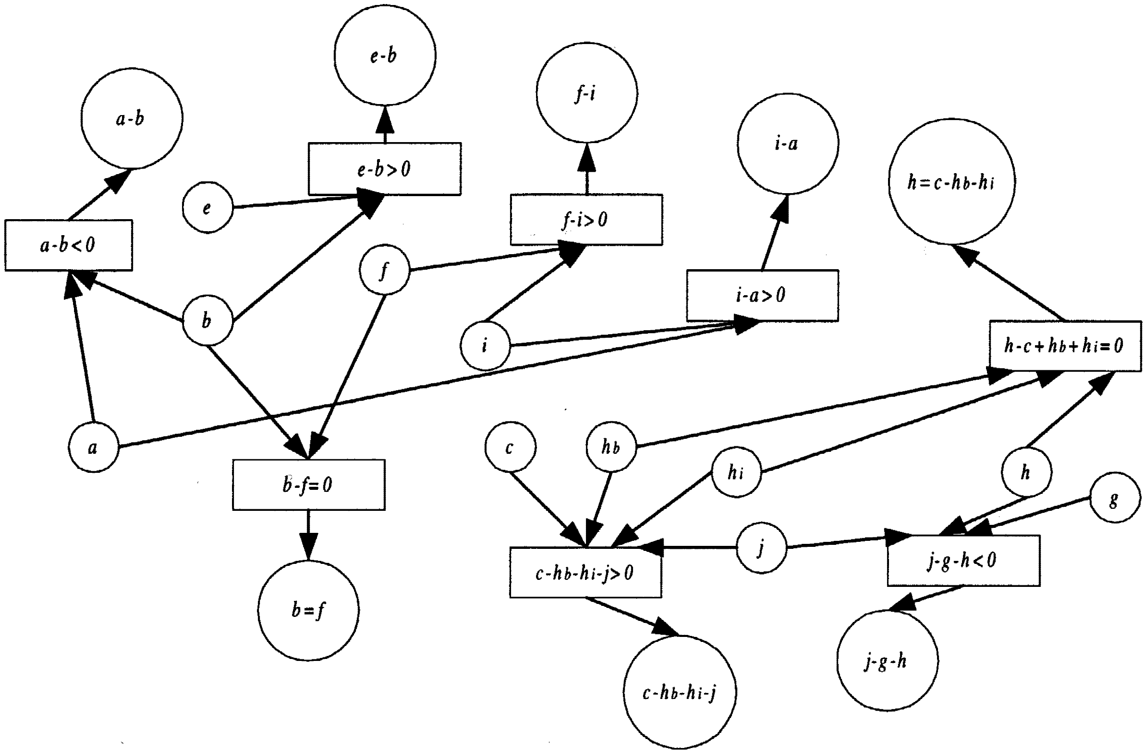


Fig. 16b. Dimension constraint network.

sentation in feature library, feature modeler, and the description of feature models.

6. GLOBAL-LOCAL DATA MODEL SCHEME

For the global-local data model scheme of the design object discussed above, the global product data model contains product characteristics or attributes that are fundamental to and shared among applications and characteristics that are passed between applications. Such a global data model is the schema of a global database. The local data model only contains product information that the application needs or completely defines products from an application viewpoint. A local data model is the schema for a local database. Product attributes for the local data model can be classified into: imported, interpreted, resident, and exported. Through classification of attributes of local data models, relationships between applications, such as data dependencies, can be formally defined (Liu, 1992).

Therefore, the global data model maintains product characteristics consistency and supports product characteristics' passing across applications, while each local data model maintains a complete product definition for an application and defines engineering roles of the application in an integrated system.

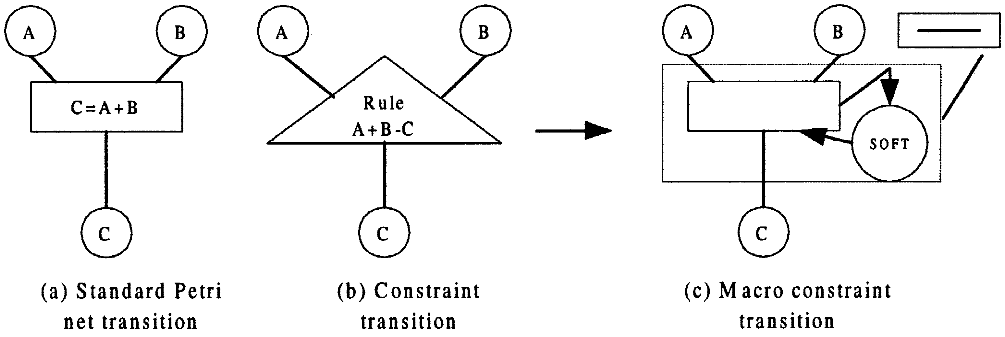

A
B
C=A+B
C
(a) Standard Petri net transition
A
B
Rule A+B-C
C
(b) Constraint transition
A
B
SOFT
C
(c) Macro constraint transition

Fig. 17. Standard, constraint, and macro constraint transition.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

6.1. Global definitions of mechanical systems and assemblies

As stated in Section 4, a mechanical system is an assembly of functional parts in which assemblies are formed by the joints between functional parts. The functional part is a part or a part assembly (subassembly) designed for mechanical functions such as transmitting or sustaining forces and torques, and made from machining or other manufacturing processes. When certain degrees of freedom (d.o.f) between parts are restricted, the parts are assembled. Hence, a part assembly is composed of parts and joints.

A part in a mechanical system is a solid entity that has specific geometry and material properties. All joint agents and functional parts, no matter whether they are parts or part assemblies, are defined on the part basis. As discussed in Section 4, geometries and features in design object model, attributes of a part in the global data model include a part identification, a name, a geometric representation, and a material property. The geometric representation of a part contains three components: geometric model, form feature, and precision features.

In the global product definition, however, a joint only defines the mating conditions and kinematic constraints between parts. From an assembling viewpoint, a joint is an ordered sequence of assembly operations and specifies assembly operations and mating conditions between parts. Based on the global-local data model scheme, a joint contains fundamental and shared characteristics of a joint from computer-aided engineering/manufacturing viewpoints, where fundamental means minimal and necessary to initiate certain applications. Hence, a joint in the global data model defines connectivity, joint agent (also a part), and kinematic degree of freedom between parts. Characteristics of a joint include an identification (id), name, joint type, connection, and degrees of freedom. A joint defines a set of mating conditions that describe the geometric mating between assembled parts and restrictions on kinematic degrees of freedom between parts, without considering the process to realize the mating condition and kinematic restrictions. According to the way that parts are assembled, a joint can be an operational joint, a fastener joint, or a fusion joint (Liu, 1992) in which a fastener joint contains additional information, that is, its joint agent(s) with a designed part (such as a pin) or a standard mechanical part (such as a bolt and nut, screw, or rivet) used as a medium to assemble parts.

6.2. Part and joint model for assembly process

Information about joints and parts of a mechanical system or an assembly in the global product definition can be used for assembly process planning, because parts are the elementary components for making an assembly and joints carry parts' connectivity information, which points to assembly features of parts. The assembly features of joints defined in the global definition indicates how parts are mated or jointed together. Both the part and joint global definition contain necessary product information for assembly process applications. However, these parts and joints require additional information such as relations (topological liaison and geometric liaison) and constraints (topological, geometric, partial precedence, stability, etc.) for assembly applications. Details were discussed in our previous work (Zha et al., 1998a, 1998b).

6.3. Product data exchange at assembly level

Product data exchange and interfaces between different CAD/CAM systems are of great importance to the integrated design environment and the future concurrent engineering and computer-integrated manufacturing systems. One possible way is to develop a suitable product model such as IGES (Initial Graphics Exchange Standard) to make the transition between different models easier; the other way is to establish an international standard such as PDES/STEP, an efficient means to increase the data compatibility of different systems.

However, current research on product data exchange is mainly limited to single-piece parts. There are no universally acceptable representation schemes for assemblies, in particular, the assembly features involved. Assembly features should be represented with different degrees of "transparency" as a computer system, while current feature concepts involved in an assembly are all joint-oriented with no consideration of various subdivisions of a machine, which are important for presenting a designer's intention on various levels. The semantics of assembly features as usually defined, therefore, should not be the same at different modeling levels. The significance of communication at assembly levels should not be underestimated. There will be an absolute need for communication of different types of data at the assembly level between real intelligent design systems in the future. There is no doubt that modeling assembled products is of increasing importance in computer-integrated systems.

The PDES/STEP standard is a neutral product model data exchange mechanism that is capable of completely representing product definition data throughout the life cycle of a product. It uses a three layer architecture, including a reference model, an object class and schema definition language—EXPRESS, and physical communication (file structure) (NIST, 1988). The product data defined in each PDES/STEP application reference model include nominal shape information (geometry, solids, and topology), form features, precision features, integration information, product structure configuration, materials, and so forth. These product data are necessary to completely define parts and part assemblies for the purpose of design, analysis, manufacturing, test, inspection, and product support. However, using only a PDES/STEP-based product specification cannot ensure integration and exchange product data at the assembly level, because interrelationships and constraints

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

between applications such as data consistency and data dependency are not defined in PDES/STEP.

Using the proposed global-local data model scheme to formalize data consistency and dependency and data passing across applications, interrelationships between applications in an integrated engineering system can be developed. Therefore, the PDES/STEP-based mechanical system product definition can be augmented to support significant portions of CAD, CAE, and CAM applications. This means that it is feasible to integrate CAD, CAE, and CAM applications by applying the global-local data model scheme as an integration model and PDES/STEP as an informal model. As described above, by use of the knowledge Petri net scheme, assembly modeling involves five aspects: function, behavior, structure, geometry, and feature. The standard interfaces at assembly levels can separate the neutral description from any specific applications or implementations through knowledge Petri net modeling, which include:

1. an English-like structured language based on Petri net for functional description;
2. standard abstraction components of dynamic systems as described in knowledge Petri net theory;
3. element mechanical connectors with functions and behaviors;
4. extension of CAD*I neutral format to assemblies (e.g., various constructive geometries and performance dimensioning of assemblies, etc.).

7. A PROTOTYPE SYSTEM FOR TOP-DOWN ASSEMBLY DESIGN AND MODELING

In principle, the new generation of CAD systems should be intelligent enough to imitate human thinking on design to some extent so as to assist designers in making decisions through the entire design process. To verify the hybrid design object model and demonstrate the effective use of it, a prototype system has been developed for top-down assembly design and modeling using knowledge-based Petri net modeling and object-oriented programming (OOP) techniques.

Coded by C/C++, Visual Prolog, and CLIPS expert system development shell with its fuzzy extension, the system incorporated an embedded CLIPS expert system shell, a knowledge Petri net tool, a function-behavior-structure modeler, a geometric solid modeler, a feature-based modeler, and a case-based reasoner. The system is therefore a prototype expert CAD system which can achieve assembly design and modeling from functional and technological specifications or customer's requirements. Figure 18 depicts the architecture of intelligent assembly design and modeling system. The output of the system is individual components and assemblies or product models, which can be used for assembly process planning and assemblability evaluation. The system can also accept the imported CAD files of individual components and assemblies from DXF and STEP based modeling system, and organize them into an assembly representation. Using feature recognition techniques,

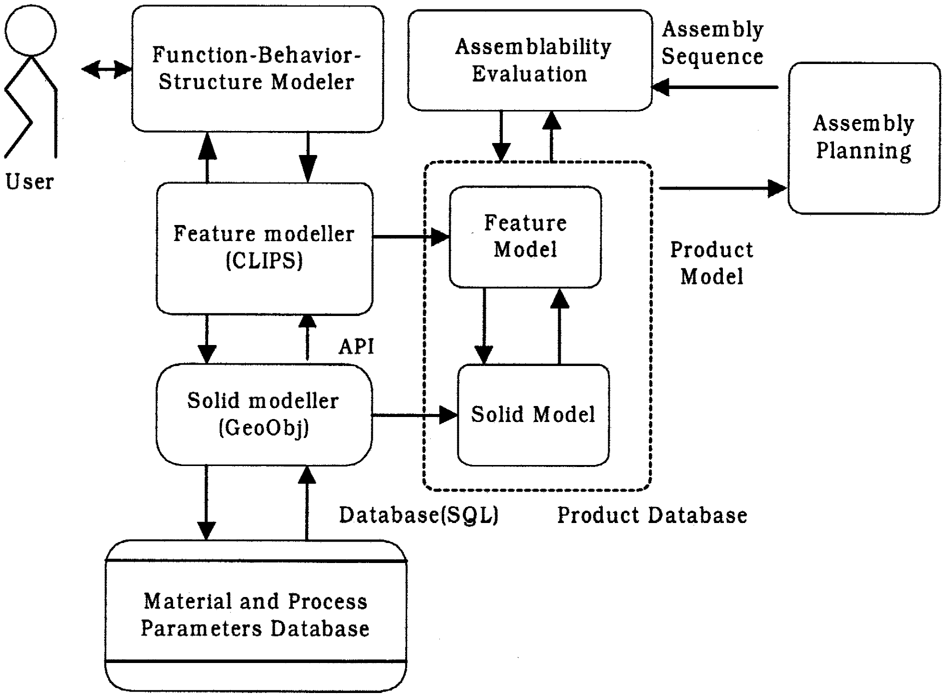


Fig. 18. The overall architecture of intelligent assembly design and modeling system.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

the assembly editor can differentiate joints between parts and assembly features on individual parts.

With this system, a number of concepts developed in the previous sections have been tested. The place-transition and knowledge Petri net models for part-joint machine description and designer's intents modeling, because of their simplicity and high abstraction, find a friendly host in knowledge-based design and modeling. Design features are consistently added for detailed design. Design alternatives of the same functional units can now be stored so that a practical intelligent modeling system can be realized. This provides the possibility of integrating a development team to work over a network, and allowing the designer to specify, design, and analyze complex mechanical systems and assemblies concurrently and cooperatively.

8. DISCUSSIONS AND CONCLUSIONS

This paper presented a new hybrid design object model for top-down assembly design. Knowledge Petri net-based design with objects scheme was utilized to uniformly model a mechanical system or an assembly and its design process. The hybrid design object model was represented and evaluated in terms of a four-level hierarchy: function-behavior, structure, geometry, and feature. The structure model is described as a place-transition based component-connector or part-joint multilevel hierarchical graph, while the functions, behaviors, geometries, features, and constraints are embedded as objects in such a hierarchy, and their causal relations are described by the corresponding knowledge Petri net graphs.

The knowledge Petri nets as a graphical language and a new knowledge representation scheme can be used to express, define, or specify, and simulate assembly design process in an interactive and integrated way. The formalisms, structures, and behaviors offered by knowledge Petri nets allow the designer to not only model mechanical systems and assemblies from function-behavior-structure description but also to manipulate and verify the assembly and its design process in different ways. Both qualitative and quantitative models are available for the knowledge Petri net assembly model. The static and dynamic characteristics in the design of assembly can be captured. The description by this model is built on more abstract concepts from assembly level to feature-based single-part level so that it can well match top-down design.

Therefore, the proposed hybrid design object model can provide a mechanism for combining device ontology and process ontology such that the designer's thinking on various levels from conceptual design to detailed design is easily captured, and allows designers to deal with incomplete, imprecise knowledge and uncertainty with the help of fuzzy logic. Such a hybrid model is evolutionary in the course of modeling and design process from top down, and has the following advantages:

1. capable of representing any machine structure;
2. capable of supporting data abstraction on any assembly modeling level;
3. captures the nature of top-down design;
4. provides a hierarchy for function-behavior-structure modeling and design;
5. is easily computerized using object-oriented programming methodology;
6. is a statically and dynamically hybrid intelligent model;
7. supports fuzzy knowledge representation and reasoning and learning; and
8. can be mathematically defined and operated by Petri net theory.

As a result, the proposed hybrid design object model using knowledge intensive Petri net formalisms makes it possible to consider all the relevant aspects in an integrated knowledge-based modeling for mechanical systems and assemblies including the up-to-date CAD technology, knowledge-based system techniques, concurrent engineering, and collaborative engineering. However, there are still some limitations in the proposed methodology and system, such as:

1. the properties analysis for the knowledge Petri nets in hybrid design object model and the mapping algorithm from function model to structure model;
2. the development and refinement of modeling system; and
3. the thorough testing of the concept of the mechanical design prototype and the establishment of a real case-based reasoning system.

Future work on this research will be required with priority on overcoming the limitations listed above, and the focus will be on the computer implementation of the proposed model.

ACKNOWLEDGMENTS

The authors would like to express their gratitude to the editor, Prof. William Birmingham and the anonymous reviewers of this paper for their valuable comments and suggestions.

REFERENCES

Berthelot, G. (1986). Transformations and decompositions of nets. In *Advances in Petri nets 1986*, Lecture Notes in Computer Science 254, (G. Rozenberg, Ed.), pp. 359–376. New York: Springer-Verlag.

Chan, S.-M., Ke, J.S., & Chang, J.P. (1990). Knowledge representation using fuzzy Petri net. *IEEE Transaction on Knowledge and Data Engineering 2*, 311–319.

Cortadella, J., Kishinevsky, M., Lavagno, L., & Yakovlev, A. (1995). Synthesizing Petri nets from state-based models, Technical Report UPC-DAC-95-09, University Politecnica de Catalunya, Spain.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

Delchambre, A. (1992). *Computer-aided Assembly Planning*. London: Chapman and Hall.

Deng, Y.M., Britton, G.A., & Tor, S.B. (1998). Design perspective of mechanical function and its object-orientated representation scheme. *Engineering with Computers 14(4)*, 309–320.

Dixon, J.R., Libradi, E.C. Jr, & Nielsen, E.H. (1990), Unsolved research issues in development of design with features systems. *Geometric Modeling for Product Engineering*, (Wozny, M.J. et al., Eds.), pp. 183–196. Amsterdam: North Holland.

Ermer, G., Goodman, E., Hawkins, R., McDowell, J., Rosenberg, R., & Sticklen, J. (1993). Steps toward integrating function-based models and bon-graphs for conceptual design in engineering. DSC-47, *Automated Modeling for Design*, ASME.

Garg, M.L., Ahson, S.I., & Gupta, P.V. (1991). A fuzzy Petri net for knowledge representation and reasoning. *Information Processing Letter 39*, 165–171.

Giacometti, F., & Chang, T.C. (1990). A framework to model parts, assemblies, and tolerances, *Proceedings of Advances in Integrated Product Design and Manufacturing, Winter Annual Meeting of the American Society of Mechanical Engineers*, Dallas, Texas, Nov. 25–30, pp. 117–125.

Groppetti, R., Santucci, A., & Senin, N. (1994). On the application of coloured Petri nets to computer aided assembly planning. *IEEE Symposium on Emerging Technologies and Factory Automation*, 381–387.

Gu, P., & Yan, X. (1996). CAD-directed automatic assembly sequence planning. *International Journal of Production Research*, 3069–3100.

Gui, J.K. (1993). *Methodology for Modelling Complete Product Assemblies*. Ph.D. Dissertation, Helsinki University of Technology

Gui, J.K., & Mantyla, M. (1994). Functional understanding of assembly modelling. *Computer-Aided Design 26(6)*, 435–451.

Heenskerk, C.J.M., & Van Luttervelt, C.A. (1989). The use of heuristics in assembly sequence planning. *Annals of the CIRP 38(1)*, 37–40.

Homem de Mello, L.S. (1989). *Task Sequence Planning for Robotic Assembly*. Ph.D. Thesis, Carnegie Mellon University, USA.

Huang,Y.F., & Lee, C.S.G.,(1989). Precedence knowledge in feature mating operation assembly planning. *Proc. of the 1989 IEEE International Conference On Robotics and Automation*, Scottsdale, Arizona, May 14–19, pp. 216-221.

Javor, A. (1995). Petri nets and AI in modeling and simulation, *Mathematics and Computers in Simulation 39(5–6)*, 477–484.

Kim, S.H., & Lee, K. (1989). An assembly modelling system for dynamic and kinematics analysis, *Computer-aided Design, 21(1)*, 2–12.

Krause, F.L., Kimura, K., Kjellberg, T., & Lu, S.C.Y. (1993). Product modelling. *Annals of the CIRP 42(2)*.

Lee, K., & Favrel, J. (1985). Hierarchical reduction method for analysis and decomposition of Petri nets. *IEEE Trans. on Systems, Man, and Cybernetics 15(2)*, 272–281.

Lee, K., & Gossard, D.C. (1985). A hierarchical data structure for representing assemblies: Part 1, *Computer-aided Design 17(1)*, 15–19.

Liebermann, L.I., & Wesley, M.A. (1977). AUTOPASS: An automatic programming system for computer controlled mechanical assembly. *IBM Journal of Research and Development 21(4)*, 321–333.

Liu, C.R., & Glaser, I. (1985). The construct of a high-level computer language for programmable assembly. *Computer and Industrial Engineering 9(3)*, 203–214.

Liu, T.H. (1992). *An Object-Oriented Assembly Applications Methodology for PDES/STEP Based Mechanical Systems*, Ph.D. Thesis, The University of Iowa.

Liu, Y., & Popplestone, R.J. (1989). Planning for assembly from solid models. *IEEE International Conference On Robotics and Automation*, Scottsdale, Arizona, May 14–19, pp. 222–227.

Looney, C.G. (1988). Fuzzy Petri net for rule-based decision making. *IEEE Transactions on Systems, Man, and Cybernetics 14*, 178–183.

Mantripragada, R., & Whitney, D.E. (1998a). Modeling and controlling variation in mechanical assemblies using state transition models, *Proceedings—IEEE International Conference on Robotics and Automation*, pp. 219–226.

Mantripragada, R., & Whitney, D.E. (1998b). Datum flow chain: A systematic approach to assembly design and modeling. *Research in Engineering Design—Theory Applications & Concurrent Engineering 10(3)*, 150–165.

Mantyla, M. (1990). A modeling system for top-down design of assembled products. *IBM Journal of Research and Development 34(5)*, 636–659.

Nieminen, J., Kanerva, J., & Mantyla, M. (1989). Feature-based design of joints. In *Advanced Geometric Modelling for Engineering Applications* (Krause, F.L. and Jansen, H., Ed.), Berlin, Germany.

NIST. (1988). *Product Data Exchange Specification* First Working Draft. Gaithersburg, MD: U.S. Department of Commerce, National Institute of Standard and Technology.

O'Grady, P., & Liang, W.Y. (1998). An object oriented approach to design with modules. Iowa Internet Laboratory Technical Report TR98-04.

Peterson, J.L. (1981). *Petri Nets Theory and The Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall.

Petri, C.A. (1962). *Kommunikation mit automaten*. Bonn, Institut fur Instrumentelle Mathematik, Schriften des IIM Nr. 2.

Popplestone, R.J., Ambler, A.P., & Bello, I.M. (1978a). RAPT: A language for describing assemblies. *The Industrial Robot 5(3)*, 131–137.

Popplestone, R.J., Ambler, A.P., & Bello, I.M. (1978b). An interpreter for a language for describing assemblies. *Artificial Intelligence 14(1)*, 79–107.

Rimscha, M.V. (1989). Feature modelling and assembly modelling—a unified approach. In *Advanced Geometric Modelling for Engineering Applications*. (Krause, F.L. and Jansen, H. Eds.), Berlin, Germany.

Rocheleau, D.N., & Lee, K. (1987). System for interactive assembly modelling. *Computer-Aided Design 19(2)*, 65–72.

Roy, U., & Liu, C.R. (1989). Establishment of functional relationships between product components in assembly database. *Computer-aided Design 20(10)*, 570–580.

Shah, J.J., & Mantyla, M. (1995). *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*. New York: Wiley.

Shah, J.J., & Rogers, M. (1993). Assembly modeling as an extension of feature-based design. *Research in Engineering Design 5*, 218–237.

Suzuki, I., & Murata, T. (1983). A method for stepwise refinement and abstraction of Petri nets. *Journal of Computer and System Sciences 27*, 51–76.

Takase, H., & Nakajima, N. (1985). A language for describing assembled machines. In *Design and Synthesis*, pp. 471–476. Amsterdam: North Holland.

Thomas, J.P., Nissanke, N., & Baker, K.D. (1996). Boundary models for assembly knowledge representation, *IEEE Transactions on Robotics and Automation 12(2)*, 302–312.

Valette, R. (1979). Analysis of Petri nets by stepwise refinements. *Journal of Computer and System Sciences 18*, 35–46.

Valette, R., Julia, S., & Tazza, M. (1994). Analysis of the behavior of a manufacturing cell with cyclic feeding policies. *Proc. 1994 IEEE International Conference on Systems, Man and Cybernetics*, San Antonio, Texas, USA, October 2–5, pp. 1683–1688.

Whitney, D.E., Mantripragada, R., Adams, J.D., & Rhee, S.J. (1999). Toward a theory for design of kinematically constrained mechanical assemblies. *International Journal of Robotics Research 18(12)*, 1235–1248.

Wilson, R.H. (1992). *On Geometric Assembly Planning*. Ph.D. Thesis, Stanford University.

Wilson, R.H., & Rit, J.F. (1991). Maintaining geometric dependencies in an assembly planning. In *Computer-Aided Mechanical Assembly Planning* (L.S. Homem de Mello, and S. Lee Eds.), pp. 217–241. Kluwer Academic Publishers.

Wolter, J.D. (1988). *On the Automatic Generation of Plans for Mechanical Assembly*. Ph.D. Thesis, The University of Michigan.

Zha, X.F. (1997). Intelligent design and assembly planning. Research Report, School of Mechanical and Production Engineering, Nanyang Technological University, Singapore.

Zha, X.F. (1999). *Knowledge Intensive Methodology for Intelligent Design and Planning of Assemblies*. Ph.D. Thesis, Nanyang Technological University, Singapore.

Zha, X.F., Lim, S.Y.E., & Fok, S.C. (1998a). Integrated knowledge-based assembly sequence planning, *International Journal of Advanced Manufacturing Technology 14(1)*, 50–64.

Zha, X.F., Lim, S.Y.E., & Fok, S.C. (1998b). Integrated intelligent design and assembly planning: A survey. *International Journal of Advanced Manufacturing Technology 14(3)*.

Zha, X.F., Lim, S.Y.E., & Fok, S.C. (1998c). Integrated knowledge Petri net based intelligent flexible assembly planning. *Journal of Intelligent Manufacturing 9(3)*.

Zhou, M.C., & DiCesare, F. (1989). Adaptive design of Petri net controllers for error recovery in automated manufacturing systems. *IEEE Trans. on Systems, Man, and Cybernetics 19(5)*, 963–973.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press

XuanFang Zha is currently a Research Fellow at School of Mechanical and Production Engineering, Nanyang Technological University, Singapore. He obtained his BEng, MEng and PhD in 1988, 1991, and 1999 respectively. His research interests include Intelligent Manufacturing System, Robotics, CAD/CAM, AI in Product Design and Manufacturing, Hybrid Intelligent Petri Nets, Micro-Elecro-Mechanical System (MEMS) Design and Simulation, Concurrent Design, Virtual Reality as well as Web-Based Design and Manufacturing.

Hejun Du is currently an associate professor at School of Mechanical and Production Engineering, Nanyang Technological University, Singapore. He obtained his BEng and MEng from Nanjing University of Aeronautics and Astronautics, Nanjing, China in 1983 and 1986 respectively. He obtained his PhD from Imperial College of Science, Technology & Medicine, London, U.K. in 1991. Since then, he has been working at School of Mechanical and Production Engineering, Nanyang Technological University, Singapore. His research interests include Micro-electro-mechanical system (MEMS), topology optimization design, concurrent design as well as smart materials and structures.

https://doi.org/10.1017/S0890060401152066 Published online by Cambridge University Press