8 Object-Oriented Techniques and Automated Methods for Robotic Assembly in Manufacturing Systems

Samuel Pierre
École Polytechnique Montréal

Monjy Rabemanantsoa
École Polytechnique Montréal

Wilfried G. Probst
Université du Québec

8.1 Introduction

Object-oriented techniques and automated methods are used in various situations for improving characterization and solutions of certain types of engineering problems related to manufacturing systems. Conceptual assembly, particularly robotic assembly, are fields where these techniques and methods are widely applied [8, 11].

Conceptual assembly is usually a complex task involving geometric and physical constraints between components which requires a large amount of information and computation. A mechanical assembly is a set of interconnected parts representing a stable unit in which each part is a solid object. Surface contacts between parts reduce the degrees of freedom for relative motion. A subassembly is a non-empty subset of these parts, having one or more elements in which every part has at least one surface contact [24, 25].

In most industrial organizations, the construction of a mechanical assembly is typically achieved by a series of assembly operations (i.e., the insertion of a bolt into a hole). The first stage in programming an assembly system is to identify the operations necessary to manufacture the given assembly and to specify the sequence in which they are to be performed. The generation of such an ordered sequence of operations is called the *assembly planning problem* [20].

The problem of robotic assembly planning can be viewed as a problem of generating assembly sequences dedicated to computer-integrated manufacturing. In fact, most computer-integrated manufacturing systems develop structures based on technical criteria for acquiring and processing data and for determining how well those systems support design documentation requirements. A flexible robotic assembly is often achieved by integrating both data processing and knowledge processing for use by manufacturing engineers in offline programming from an assembly plan, which is an ordered sequence of operations that constructs the product from its component parts [25]. The determination of a plan involves searching in a state space where a feasible course of action would transfer the parts from their initial states to goal states.

There exists in the literature various approaches to performing assembly planning [27, 28]. Some of these generate only monotone and linear plans while others generate monotone and non-linear plans. Other plans deal with non-monotone and non-linear assembly planning using the technique of *ray tracing* in which parts are represented by the Constructive Solid Geometry (CSG) model [15, 30]. A plan is linear if each operation performed in it moves only one part at a time. In cases where groups of parts can be moved together, plans are considered to be non-linear and are referred to as *subassemblies.*

Considering certain combinatorial aspects related to plan generation, and hence to robotic assembly, many researchers have proposed the use of heuristic-search techniques and intelligent-automated methods to deal with robotic assembly in manufacturing systems. This chapter follows this orientation and presents various object-oriented techniques, which perform robotic assembly in manufacturing systems. It is organized as follows: Section 8.2 summarizes the fundamentals of robotic assembly and discusses some robotic systems. Section 8.3 expounds on the basic principles of object-oriented techniques. Section 8.4 analyzes certain robotic and automated assembly architectures. Finally, Section 8.5 describes some automated assembly applications.

8.2 Intelligent Assembly Planning and Knowledge Representation

Planning is a process that searches for a sequence of actions (or operations) in order to achieve a goal statement. This original objective is normally too complex or too abstract to be accomplished through a single operation, so it is often necessary to break it down into smaller, simpler subgoals. In an ideal situation, a subplan for each subgoal can be formulated independently and a complete plan can be derived by simply combining the subplans. However, in practice, subplans often interact (i.e., the achievement of certain goals may actually prevent the accomplishment of others). This conflict problem presents major difficulties in a planning process but Artificial Intelligence techniques may be helpful in addressing such difficulties [5, 16, 29].

Structure of an Intelligent Assembly Planning System

An intelligent assembly planning system can be structured as a typical knowledge-based system [3]. As shown in Fig. 8.1, it contains a knowledge base, a control structure, and a blackboard. The knowledge base records information about the assembly problem domain and the expertise of the assembly planning. It includes workpiece structures, assembly operations, and assembly principles.

The workpiece structures describe the relationships among workpiece components and their associated properties; whereas, the assembly operations are descriptions of robot actions that can be used to assemble workpieces. The assembly principles can be viewed as "rules of thumb," (i.e., empirical guidelines of assembly experts), including the methods of ordering the component assembly sequence and posting constraints should certain situations arise [5]. The control structure utilizes knowledge-base information to generate assembly plans and integrates a structure analyzer and a plan generator.

As shown in Fig. 8.2, the planning process consists of two phases: structure analysis and plan generation [5]. During structure analysis, the system analyzes the workpiece structures according to the guidelines of assembly principles. This includes: refining an abstract workpiece (one that can be disassembled into

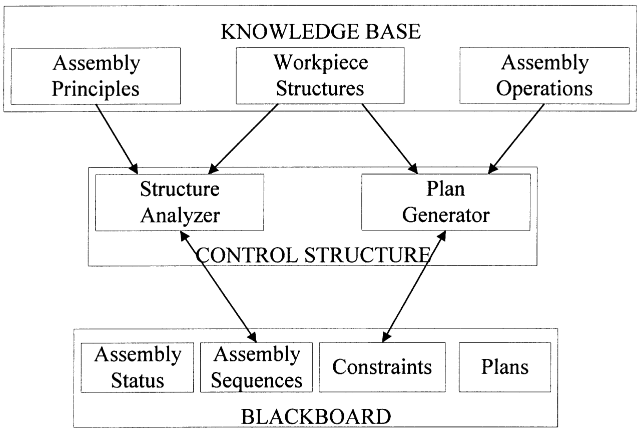


FIGURE 8.1 Components of a Mechanical Assembly Planning System.

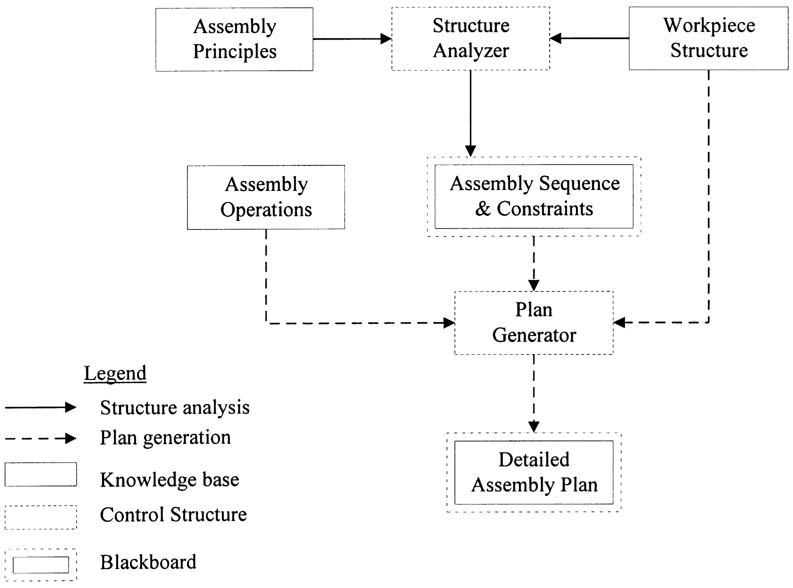


FIGURE 8.2 Different Phases of a Planning Process.

more than one part) into sub-components; deciding on the subcomponent assembly sequence; and posting assembly operation constraints for each subcomponent. A plan generator subsequently follows analysis information in order to select assembly operations. The status of the assembly domain, the properties of the components, and the constraints determine the selection of operations.

Finally, the blackboard records intermediate hypotheses and decisions, which the system manipulates. This includes: the current state of the assembly or assembly status; components being assembled or assembly sequences; and constraints and plans.

Knowledge Representation in Intelligent Robotic Systems

The area of robotics and automation systems has been undergoing an expanded revolution over the past three decades. At first, computing and sensory capabilities were not present in robotic systems. The first generation evolved into systems with limited computational and feedback capabilities. The second generation began including systems with multi-sensory and decision-making capabilities. The third generation robotic systems, called intelligent robotic systems, are equipped with a diverse set of visual and non-visual sensors and are being designed to adapt to changes within their workspace environment [23].

Regarding intelligent robotic systems, their modeling requires utilization and implementation of concepts and ideas drawn from various fields. In point of fact, Artificial Intelligence and, more precisely, knowledge-based systems as well as Operations Research and Control System Theory, constitute the background required to tackle the multiple challenges of such a modeling. Industrial automation systems, Flexible Manufacturing Systems (FMS), Computer Integrated Manufacturing (CIM) systems in general as well as telerobotic systems, are among the more frequent applications [24].

In intelligent robotic systems, knowledge may be represented either symbolically or numerically. Symbolic knowledge representation includes: rule-based, frame-based, associative networks, logic-based, and object-oriented systems. Numerical representation refers to mathematical models and is contingent upon the specific application domain.

For intelligent robotic and automation systems, time may or may not be critical in knowledge representation techniques depending on the level of knowledge processing. For example, the time factor is critical in knowledge-based systems for control applications as most of these have been developed to enhance, monitor, or modify online the dynamic system operation. Generally, an offline knowledge-based system may be developed for high level planning and decision making and a real-time system for online information processing. However, overall timing constraints may have to be taken into account regarding the specific application under consideration [32].

At least two criteria are frequently used to compare and judge knowledge representation schemes for the particular types of systems [14]:

1. Expressive power of the representation in comparison with other representation schemes and the range of control problems for which the representation is suitable; and
2. Computational complexity of reasoning, using particular representations, in comparison with other representations and the limitations of its applicability imposed by issues of complexity.

Recently, the topic of intelligent robots has attracted a great deal of attention among researchers. Considering the social situation of the lack of highly skilled, experienced technicians and workers, it would seem logical to automate the manufacturing and fabricating process. Proceeding from this idea, Kamrani et al. [13] have proposed an intelligent knowledge-based system as a solution to the problem of selecting an optimal robot for cell design. New tasks are being defined for robots in order to meet the challenges of flexible manufacturing systems. Associated with this is an increasing variety of robots from which to choose. The selection of an optimal robot for a particular task constitutes a major problem.

Various parameters must be considered and the user should choose an industrial robot whose characteristics satisfy the requirements of the intended tasks. Massay et al. [17] have proposed a hierarchical design approach, which can serve as a basis for the off-line development of effective robotic systems.

On the other hand, a hardware and software methodology leading to a knowledge-based system has been introduced in [32] for use in intelligent robotic systems. This knowledge-based system has been derived for the organizational level of such a system and is being used to develop off-line plan scenarios to execute a user-requested job. The knowledge base contains all the information related to the class of problems that the system has been designed to solve, while the inference engine operates upon the knowledge base. Although separate entities, the knowledge base and inference engine have been so designed as to enable them to operate closely together as a unit.

Similarly, an expert support system has been proposed to provide flexible manufacturing systems (FMS) with the capability of adjusting in real-time to changes in the manufacturing environment. The key component of this support system is the "information cell," which is controlled by the flow of information between the cell and its environment.

The concept and prototype of a hierarchically structured knowledge-based system for coordinated control of a welding robot and a positioning table is presented in [31]. The knowledge-based system determines appropriate weld parameters (voltage, speed, feedrate) on the basis of the job description. It is also capable of planning the optimum table orientation and robot trajectory and controlling the welding process [19].

An integrated environment for intelligent manufacturing automation has been proposed in [3]. The knowledge-based integrated environment is based on the following components: interface to external environment, meta-knowledge base, database, inference mechanism, static blackboard, and interface to other subsystems. This approach has been applied to the product design in a manufacturing process.

Planning a sequence of robot actions is especially difficult when the outcome of actions is uncertain, as is inevitable when interacting with the physical environment. Christiansen and Golbert [6] have compared two algorithms for automatic planning by robots in stochastic environments: an exponential-time algorithm maximizing probability and a polynomial-time algorithm maximizing a lower bound on the probability. They have considered the case of finite state and action spaces where actions can be modeled as Markov transitions. As these algorithms trade off plan time for plan quality, their performance is compared to a mechanical system for orienting parts. This leads to two properties of stochastic actions which can be used to choose between these planning algorithms for other applications.

Regardless of specific applications, the knowledge-based systems described above have been developed to enhance the capabilities and flexibility of robotic systems dedicated to industrial automation. Their common limitation is that they are not easily modifiable to accommodate different versions or potential deviations from the initial class of problems for which they have been designed [32]. Object-oriented techniques integrating concepts, such as class, instance, inheritance, and genericity, can deal with such a limitation.

8.3 Principles of Object-Oriented Techniques

Object-Oriented (OO) technology constitutes one of the most important software evolutions of the 1990s. It refers to object-oriented programming, design, analysis, and databases, and covers methods for either design or system analysis. An overview of such a technology is presented in this section.

Basic Concepts and Definitions

In the field of object-oriented techniques, designers think in terms of objects: they create objects; add behaviour to them; make them interact; and observe the results. From a programming point of view, an *object* represents anything real or abstract (whose attributes are represented by data types and behaviors) which are controlled by operations. An object is defined by a list of abstract attributes often called *instances* or *class variables.* Communication between objects is done across well-defined *interfaces.*

Figure 8.3 illustrates an object, whereas Fig. 8.4 gives an object-oriented example of an “Employee.” Moreover, *objects* can be categorized into object types, which is specified during object-oriented analysis. *Classes* refer to the software implementation of such object types. Details of classes are determined in object-oriented design.

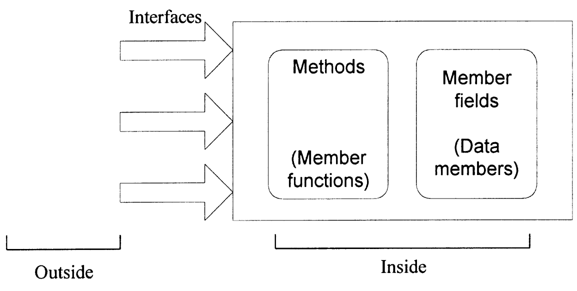


FIGURE 8.3 Concept of an Object.

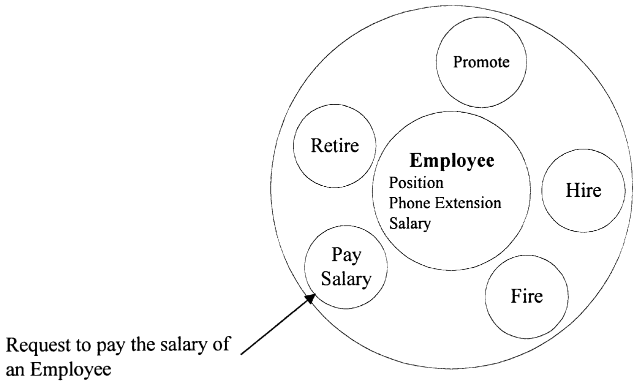


FIGURE 8.4 An Example of an Object.

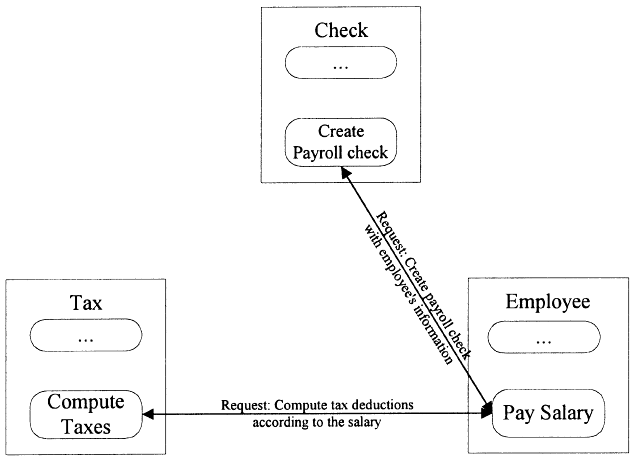

Check
...
Create Payroll check
Request: Create payroll check with employee's information
Tax
...
Compute Taxes
Employee
...
Pay Salary
Request: Compute tax deductions according to the salary

FIGURE 8.5 Messages Passing among Classes.

In order to interact with an object, a request is sent to it, causing an operation to be triggered. Such operations are called *member functions* or *methods* in object-oriented programming and are elicited from responses to messages transmitted to objects. Therefore, a method constitutes a procedural specification that alters the state of an object or determines the message to be successfully processed by such an object.

Fig. 8.5 illustrates the message passing between three classes of objects. Concretely, a class represents a collection of objects sharing attributes and methods.

Encapsulation refers to the practice of including everything within an object that it requires, in such a way, that no other object ever needs to be aware of its internal structure which resulted in packaging together its data and operations. Consequently, details of its implementation are hidden from its user. This is referred to as *information hiding*, in that the object conceals its data from other objects and allows the data to be accessed via its own operations.

An *abstract data type* (ADT) is an abstraction similar to a class that describes a set of objects in terms of an encapsulated or hidden data structure. All interfacing occurs through operations defined within the ADT, providing a well-defined means for accessing objects of a data type and running the appropriate method. Therefore, although objects know how to communicate with one another across well-defined interfaces, they are not normally allowed to know how other objects are implemented, thus protecting their data from arbitrary and unintended use. To summarize, the ADT gives objects a public interface through its permitted operations. Furthermore, all operations that apply to a particular ADT also apply to the instances of that ADT.

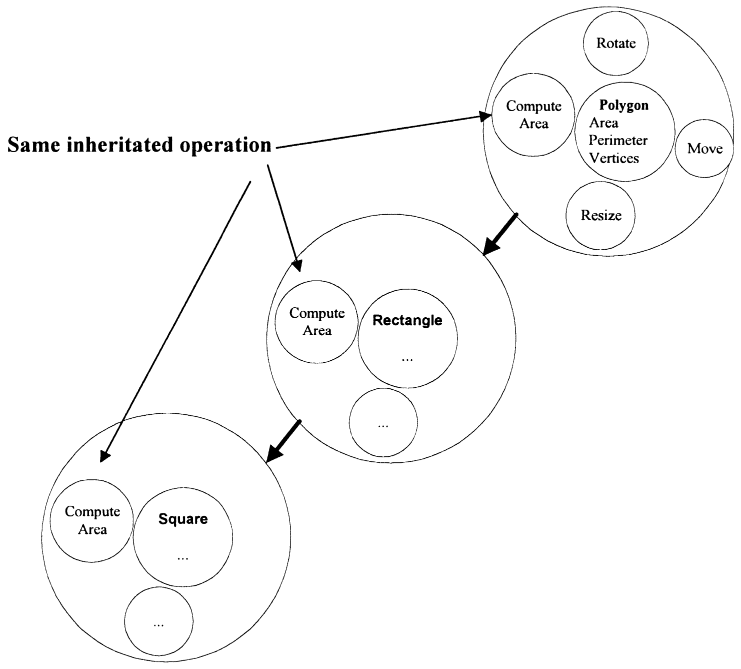


FIGURE 8.6 Example of Inheritance.

Furthermore, deriving new classes from existing classes is called *class inheritance* (or simply *inheritance*) and constitutes an implementation of generalization. The latter states that the properties of an object type apply to its subtypes which have the derived classes. With single inheritance, a class is derived from a one-base class.

In multiple inheritance, a class can inherit data structures and operations from more than one superclass. Fig. 8.6 illustrates a simple inheritance, where the class *Rectangle* inherits several operations from *Polygon* and *Square* inherits certain operations from *Rectangle*. The real strength of inheritance comes from the ability to make data structures and operations of a class physically available for reuse by its subclasses.

Polymorphism refers to the ability of objects, belonging to different classes but related by inheritance, to respond dissimilarly to the same member function call. It enables the writing of programs in a general fashion in order to handle a wide variety of existing and specified related classes. One strength of polymorphism is that a request for an operation can be made without knowing which method should be invoked. It is particularly effective for implementing layered software systems. For instance, in operating systems, each type of physical device may operate quite differently from the others, whereas commands to read or to write data from and to those devices can have a measure of uniformity.

Therefore, object-oriented techniques provide facilities for representing two impressive concepts: abstraction and inheritance. They also change the way designers or developers think about systems. In fact, this way of thinking is more natural for most people than the techniques of structured analysis and design. As well, all entities consist of objects that have certain types of behavior. Certain objects may be quite different despite sharing many of the same attributes and exhibiting similar behaviors.

Special Features

In the domain of object-oriented techniques, analysts, designers, and programmers use the same conceptual models for representing object types and their behaviors. They all draw hierarchies of such object types where the subtypes share the properties of their parent. Also, they consider objects as being composed of other objects, use generalization or encapsulation, and think about events changing the states of objects and triggering certain operations.

Program complexity is a measure of program understandability. Avoiding such complexity improves program reliability and reduces the effort needed for its development and maintenance. Object-oriented techniques provide two important ways to divide complex software into simple procedures. First, methods result in a state change of an object which is usually simple and easy to implement. Secondly, each operation is isolated from cause and effect. In fact, after being triggered by a number of events, an operation executes the associated method to change the state of an object but does not recognize its cause and effect. That is to say, the operation has no knowledge of what event caused it to happen, nor why, and is unaware of what operations will be set off as a consequence of its events. Therefore, reduction in complexity is partly due to the fact that object-oriented classes can be self-contained and divided into methods.

Moreover, systems can be built from existing objects since inheriting operations from a superclass enables code sharing and structure reuse among classes. When creating a new class, instead of completely rewriting data members and member functions, the programmer can designate the new class to inherit them from a previously defined class. This leads to a high degree of reusability, which saves money, shortens development time, and increases system reliability.

Today, software factories manage well as possible libraries of reusable classes, to maximize reusability and minimize maintenance costs. Developers who create new classes are evaluated according to the re-use of such classes. On the other hand, designers who utilize existing classes are evaluated regarding the way they re-use those classes. In other words, developers or designers should strive to increase the functionality of the classes they use by building new objects from existing objects. Moreover, object types can be designed for customization according to the needs of various systems, resulting in a method that can be reused on different software platforms and on multiple interacting processors.

Object-oriented techniques allow for the building of complex and reliable systems in a simple way. The code is directly related to defined objects and to methods that manipulate those objects. Each method is relatively simple; easy to change; and can be quickly created using pictures, tables, declarative statements, equations, database access, report generation, rules, and nonprocedural techniques. As systems become increasingly complex, new functionality can be directly added to existing classes for building these powerful systems. Therefore, by using good class libraries, a designer can assemble complex software in a fluid way. There is no need to be concerned with the internal workings of a class. If an object type functions well, it can be treated as a black box whose interior never necessitates scrutiny. Furthermore, created objects can be quickly and repeatedly modified.

Obviously, object-oriented techniques result in numerous advantages. First, each object in a system turns out to be relatively small, self-contained, and manageable. This reduces the complexity of systems development and may lead to higher quality systems, which are less expensive to build and maintain. In addition, once an object is defined, implemented, and tested, it can be reused in other systems. Indeed, reusability can greatly increase productivity since the reused objects are generally proven products. Finally, an object-oriented system can be modified or enhanced very easily by changing some types of objects or by adding new types of objects without interfering with the rest of the system. Those potential benefits constitute the driving force behind the object-oriented revolution.

Methodology and Models

Object-oriented techniques enable the building of real-life models that include two important features: a representation of the object types with their structures and a representation of the object behaviors. A set of diagrams should be created for each of these aspects: object-structure diagrams showing the objects and their interrelationships, and event diagrams showing what happens to the objects. The first aspect concerns object types, relationships between objects, and inheritance. It refers to the Object Structure Analysis (OSA) and corresponds to the class structure design. The second view concerns the behavior of objects and what happens to them over time, which refers to the Object Behaviour Analysis (OBA) and method design.

The OSA defines the object types and the way in which they are associated. This leads to the following questions:

- What types of objects exist? What are their functions and how are they related? What are the useful subtypes and supertypes?
- Is a certain kind of object composed of other objects?
- Such questions allow for the identification of the classes and the definition of superclasses, subclasses, their inheritance relationships and the methods to be used, which result in the detailed design of the data structure.

The OBA relates to the following questions:

- Which states can the object types be in? What types of events change these states?
- What succession of events occurs?
- What operations result in these events and how are they triggered?
- Which rules govern the actions taken when events occur?

These questions lead to the detailed design of methods, using either procedural or nonprocedural techniques. Consequently, the input to the code generator is developed, screen design is done, dialogues between objects are designed and generated, and prototypes are built.

Events cause an object-oriented system to take various actions. In order to describe processes in terms of events, triggers, conditions, and operations, event diagrams constitute a primary means of communicating object-oriented behavior, showing events and the operations set in motion by the methods. They express processes in a more rigorous fashion which makes the code easy to be generated using a sequence of operations (drawn with rounded-cornered boxes and easily understood). Such diagrams must be precise (with an engineering-like precision), which can speed up work, improve the results, enhance creativity, and simplify maintenance. For strategic-level planning, an Object-Flow Diagram (OFD) is useful for indicating the constructed objects and the activities that produce and exchange them. A three-dimensional box is used to represent real-life objects that flow between activities.

In an object-oriented context, the data structure of an object type can be manipulated only through the methods of the object class. As for changing the state of an object, requests must be sent in order to activate the associated methods. Each state change is usually simple to program, which leads to the division of programming into relatively uncomplicated parts. Also, each object performs a specific function independently of the others, and responds to requests neither knowing the reasons for such requests nor the consequences of their actions. As a result, classes can be largely changed independent of other classes, which renders them relatively easy to test and to modify.

8.4 Applications to Robotic and Automated Assembly

In a robotic and assembly manufacturing environment, the available flexibility introduces another degree of complexity in decision making. The assembly operations are descriptions of robot actions that can be used to match components and subcomponents. It is important that robot planners not be obliged to analyze an assembly but rather to use a path-planning generated by the assembly in order to structure the issuing of task level commands. To achieve this, the architecture must be designed to retrieve data through contents and not through a fixed predefined structure. "Which components have a champfix and five edges?" is a type of question which requires an answer that object-oriented databases can satisfactorily provide.

Object-Oriented Modeling

The way object-oriented databases reason about geometry is by recognizing certain geometric features of objects [18]. To this end, representations of designed objects in terms of features must be developed. As a result, an important use of the database is the design of a knowledge-based system. Learning techniques can also be used to obtain relevant information automatically and to guide the system in search of good planning solutions [2].

As presented earlier, object-oriented concepts are based on fundamental principles of complexity management which are very helpful for complex object modeling and data manipulation related to complex entities such as computer-aided design (CAD) or computer-aided engineering (CAE) system environment [1, 10]. In the assembly domain, the objects are typically composites or aggregates of components. The term "composite object" is commonly used to denote a layered abstraction model. The use of abstract data types enhance program modularity since modifying the object structure does not affect the manipulation of external objects. Abstract data type can be implemented as an object collection with the same structure and representing a class.

Each object is characterized by a specific identifier OID (object identifier) [25]. However, two similar objects with the same dimensions, but a dissimilar OID, are said to be different. Thus, through the development phases of composite objects, the OID provides a natural paradigm for maintaining the uniqueness of objects independent of structure or content.

Object identity allows for direct graph modeling and objects are characterized by properties. A property may be an object characteristic such as an attribute, a function, or a subobject component. For example, the object *circle* can have the following properties:

- Simple attribute: radius (R);
- Composite attribute: center (x, y, z);
- Function: surface (πR^2).

A class may be defined as a description of the behavior of a collection of objects in a modular way (e.g., circles with the same radius, polygons wth a common shape). The concept of "class hierarchy" eliminates the possibility of specifying and storing redundant information. It integrates the previous notions of "superclass" as a higher level mode; "subclass" as a lower level mode; and "methods" as operations that can either retrieve or update the state of an object.

For assembly process planning, methods are very useful in an inheritance hierarchy (a method defined for a class is inherited by its subclasses). Data integrity is ensured by a procedure called "demon," a triggered operation associated with an object when a particular condition occurs. In this context, geometrical and topological information is not sufficient for establishing assembly planning design rules. Geometric interference, physical constraints, tolerances, and kinematic relationships are all important issues to be addressed in assembly planning design.

Geometry helps to make a representation scheme for parts and products, and it refers to the places where the objects are located. Topology defines the connectivity between geometric elements and is in charge of processing the rules for connecting elements of geometry to produce a part. It also refers to the reasons why objects are located in some specific places. Tolerances can be defined as acceptable variations between design and manufacturing processes that allow for the various components to be assembled into a product. As a result, the use of an object-oriented approach provides the tool, without any interactive means, for modeling entities and properties, as well as for managing design information effectively. Fig. 8.7 shows an object-oriented representation of a superclass SOLID-3D.

Knowledge Representation

Various approaches have been proposed in assembly-planning literature [1, 4, 7, 9, 28]. Rabemanantsoa and Pierre [21–23] have also proposed an approach based on the idea of viewing the planning with intermediate states and considering the whole assembly as a hierarchy of structures. Therefore, the knowledge base integrated in our system consists of three parts: component structures, intermediate states, and primitive structures. These parts contain the physical information related to knowledge of the design intent such as geometry topology, features, and tolerances. Each component is represented by a frame, which consists of a name and a number of properties. For example, Fig. 8.8 shows the panel frame of a mechanical part: number of faces, number of flat surface, composite surface, etc.

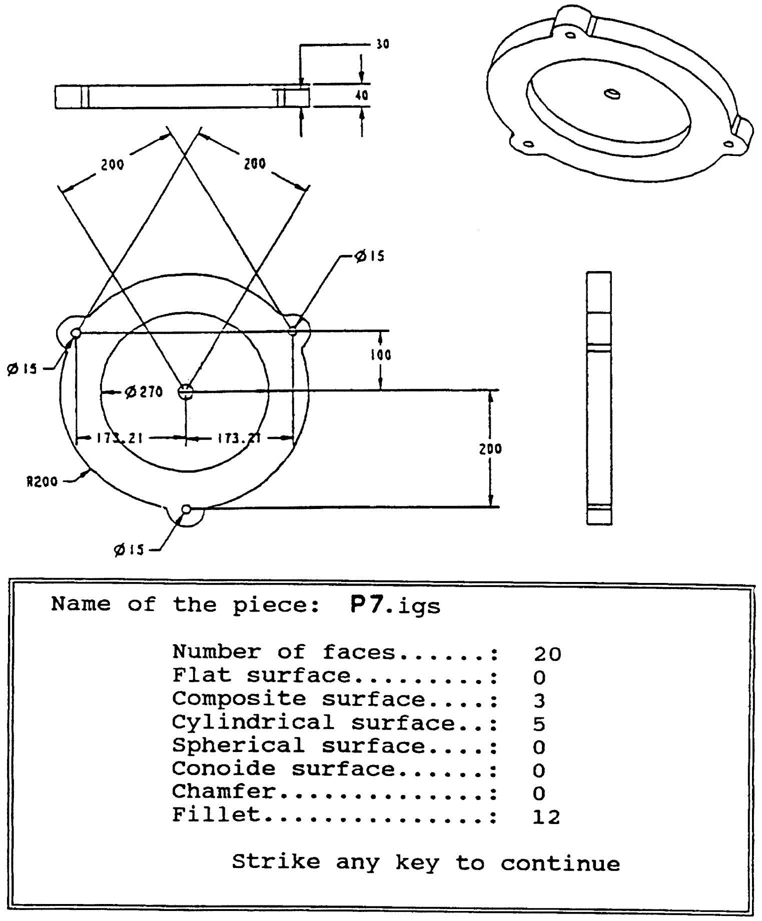
***Class**: Solid_3D*
***Inherit**: Object*
***Class structure**:*

SuperClass	*: (Sphere, Ellipsoid, Paraboloid, Cone, Cylinder, Hyperboloid)*
Class	*: (Subsphere, Subellipsoid, Subparaboloid, SubCone, Subcylinder, Subhyperboloid)*
SubClass	*: (Arcs, Edges, Vertices, Points)*

***Structure**:*

Attribute, Transformation

FIGURE 8.7 Object-Oriented Representation of a Complex Object.



```
Name of the piece:  P7.igs

        Number of faces......:  20
        Flat surface.........:  0
        Composite surface....:  3
        Cylindrical surface..:  5
        Spherical surface....:  0
        Conoide surface......:  0
        Chamfer..............:  0
        Fillet...............:  12

            Strike any key to continue
```

FIGURE 8.8 Panel of Mechanical Parts.

To characterize the intermediate states, the input scheme is represented by a set of assembly operations along with their states. In this context, the state of a component is defined as either the initial component or another position induced by a set of predefined operations [20]. Given a product p with "n" components, we have $P = (p_1, p_2,\ldots,p_n)$, and the set of states of each component is $S = (s_1, s_2,\ldots,s_k)$ for "k" maximum states.

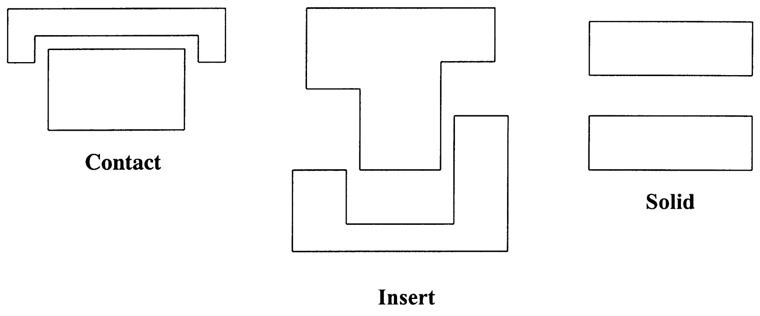


FIGURE 8.9 Primitive structures.

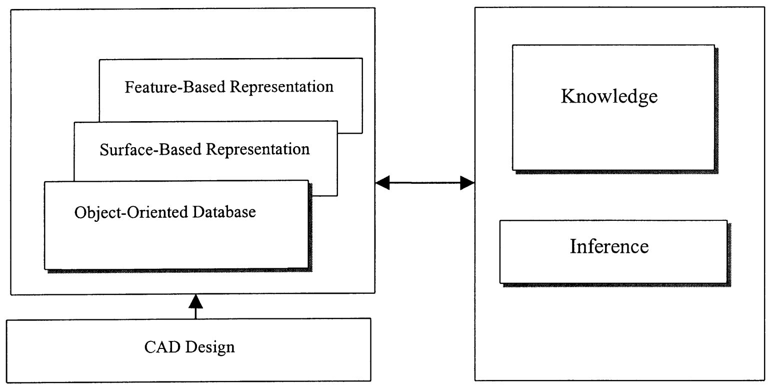


FIGURE 8.10 Database Structure.

Let the set of predefined assembly trajectories for each component in P be $T = (t_1, t_2, \ldots, t_j)$ where j is the total number of trajectories. The knowledge contains a change of state along a fixed trajectory, that is, for component $p_i \in P$, an assembly operation is denoted as a triple:

$$A = \{P, S, T\}$$

The primitive structures are necessary to ensure the completeness of the knowledge. They involve a small number of parts or components which interact positionally with one another in simple ways [21]. The three types of primitive structures currently being considered are shown in Fig. 8.9:

1. Contact: in this structure; two components interact in such a way that they are in contact and each component can be moved into the structure in any direction through an infinite half-space.
2. Insert: in this structure, two components interact in such a way that each one has to mate in a specific direction if the other is already present.
3. Solid: this structure is similar to (1), except for the two components not being in contact with each other.

Another way to consider the knowledge representation is using a CAD database. This architecture provides different levels of abstraction for modeling the parts through the object-oriented database illustrated in Fig. 8.10.

The link structure of the system between the database and the CAD design uses the method developed by Shah and Bhatnagar [27]: intrinsic and explicit characteristics (type 1); intrinsic and implicit characteristics (type 2); and extrinsic characteristics (type 3). For example, a hole should be described in terms of diameter, length, and an orientation vector; these parameters are type 1 because they may be available in the database itself. The radius of a hole is type 2 because it may be derived from the center coordinates and the diameter. However, if the angle between the axes of two holes of equal diameter is needed, this type 3 information is not available in the database itself, but would depend on the concept utilized in the model.

The Artificial Intelligence (AI) techniques relating to *decision tree* and *production rule* provide better understanding of the surfaces. For example; if two surfaces have the same ordinate, the AI techniques make it possible to check whether these surfaces are adjacent or in contact. The positions of the surfaces depend on the spatial relationships between each pair of components by using the coordinate frame (body axis system) attached to the basic component. The knowledge is expressed in terms of mobility (M) and contact functions for each pair of parts. The two functions C and M are then used by an expert system called XGEN to generate assembly sequences [25].

As defined earlier, a mechanical assembly is a composition of interconnected parts forming a stable unit. Interconnection implies that one or more surfaces are in contact. The body axis system has six directions attached to each basic component. Contact and mobility of one component with respect to another are then evaluated for each of the n components within the six degrees of freedom in a spatial relationship. This results in a combination of two into n.

As illustrated in Fig. 8.11, the contact C of component "b" with respect to component "a" is a function $C(a, b) = (V_1,V_2,V_3,V_4,V_5,V_6)$, and the relative mobility M of component "b" with respect to component "a" is a function $M(a, b) = (V_1,V_2,V_3,V_4,V_5,V_6)$. The part union angle θ is used to express the contact angle, while the rotation ϕ is used to express the relative mobility angle. A part mating along an assembly axis having $\theta_i = 0$ and $\phi_i = 0$ is defined as an orthogonal assembly trajectory; when $\theta_i \neq 0$ and $\phi_i \neq 0$, we have a non-orthogonal assembly trajectory. An attachment by means of a screw is defined as a circular assembly trajectory. Hence, contact and mobility functions are expressed as follows:

$$C(a,b) = (V_1 : \theta_1,\ V_2 : \theta_2,V_3 : \theta_3,\ V_4,V_5,V_6) \text{ and}$$

$$M(a,b) = (V_1 : \phi_x,\ V_2 : \phi_y,V_3 : \phi_z,\ V_4,V_5,V_6)$$

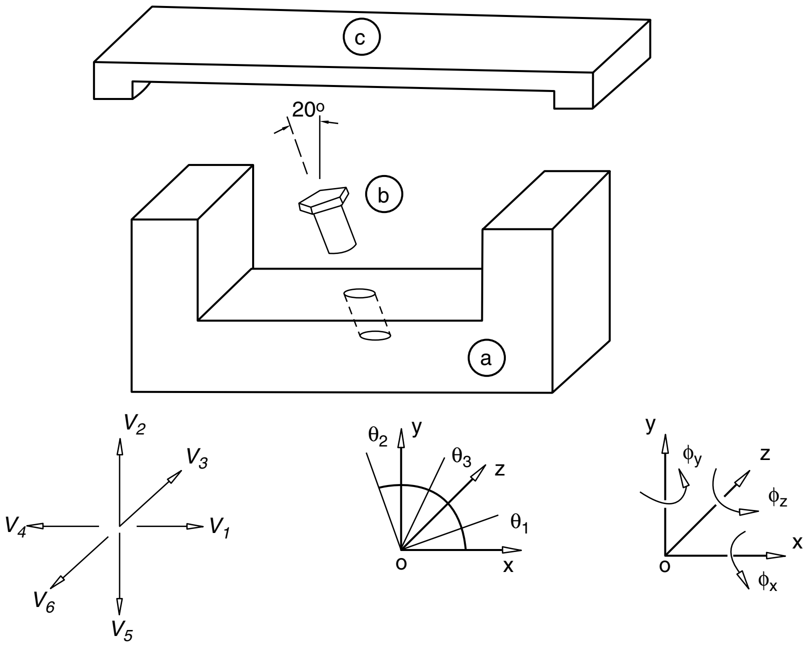


FIGURE 8.11 Example of a Non-Orthogonal Assembly Trajectory.

TABLE 8.1 Knowledge Representation

C: Contact	M: Mobility
$C(a,b) = (2,1{:}20,2,2,1,2)$	$M(a,b) = (0,4,1{:}-20,0,0,0)$
$C(a,c) = (2,0,0,2,1,0)$	$M(a,c) = (0,1,1,0,0,1)$
$C(b,c) = (0,0,0,0,0,0)$	$M(b,c) = (1,1,1,1,0,1)$
$\theta_1 = 0$; $\theta_2 = 20°$; $\theta_3 = 0$	$\phi_y = 0$; $\phi_z = -20°$
Note:	Note:
0 = no contact	0 = no relative mobility
1 = presence of contact	1 = possibility of moboility
2 = lateral contact	3 = circular trajectory
	4 = non-orthogonal trajectory

Figure 8.11 is an example of a non-orthogonal assembly trajectory developed with this system. For the set of parts composed of three components, there are $3! = 6$ possible solutions. To overcome the *combinatorial explosion* risk, our approach uses V_k ($k = 1,6$) to represent the six degrees of freedom. The spatial relationship of one component, with respect to another one, considers one pair of components, that is, the combination of 2 into n, where $n = 3$. The referential {R2} provides an "explicit relative mobility condition." Both referentials are attached to the lower component when considering each pair of components. Furthermore, from the semantic data modeling point of view, each V_k ($k = 1,6$) is coded.

In the contact function C, we have $V_k = 0,1,2$ representing the absence of contact, presence of contact, and presence of lateral contact respectively. When we take into account these mating conditions, the contact function of component "b" in relation to component "a" (Fig. 8.11) is expressed as follows: $C(a,b) = (2,1{:}20,2,2,1,2)$, in which $V_1 = V_3, = V_4 = V_6 = 2$, indicating the presence of lateral contact along these directions. $(V_2{:}\,\theta_2) = (1{:}\ 20)$ indicates the contact angle and the assembly trajectory at a positive angle equal to 20° with respect to $V_2 = Y$ axis. $V_5 = 1$ shows contact between "b" and "a" in that direction.

In the relative mobility function M, we have $V_k = 0,1,2,3,4$, representing no relative mobility, the possibility of mobility, presence of an orthogonal assembly trajectory, existence of a circular trajectory, and presence of a non-orthogonal assembly trajectory respectively. Hence, the relative mobility of component "b" in relation to component "a" is $M(a,b) = (0,4,1{:}20,0,0,0)$ in which $V_1 = V_4 = V_5, V_6 = 0$, indicating that "b" cannot be moved along these directions. $V_2 = 4$ signifies that there is a non-orthogonal assembly trajectory using $V_2 = Y$ as reference axis. $(V_3{:}\ \phi_z) = (1{:}\ -20)$ indicates that the relative mobility is at −20° (clockwise rotation) with respect to the $V_3 = Z$ axis. The knowledge representation for spaital data models, relating to the example of Fig. 8.11, is shown in Table 8.1.

Assembly Operations

The plan generation formulates the assembly operations, using the knowledge base plus a set of scheduling rules, to achieve the goal. Using learning techniques, the supervision architecture is given capabilities for generating the plan. For each plan level, the main functions consist of dispatching, monitoring, diagnosis, and recovery [21]. The result is a decision tree:

- Dispatching: taking care of global coordination activities to be performed by a high-level cell controller. Approaches to derive the control algorithm are usually based on constant rules used to post constraints. Dispatching is the basic "operation-selection."
- Monitoring: in model-based monitoring, the sensory conditions to test in each situation are well defined in the model. Moreover, discrete monitoring is used to check goal achievement after the execution of operations. Continuous monitoring is used to check sensory conditions during the execution of operations. If an exception is detected, the diagnosis function is called upon.
- Recovery: this function is called to update the global state and the complete operation description is stored in the plan.

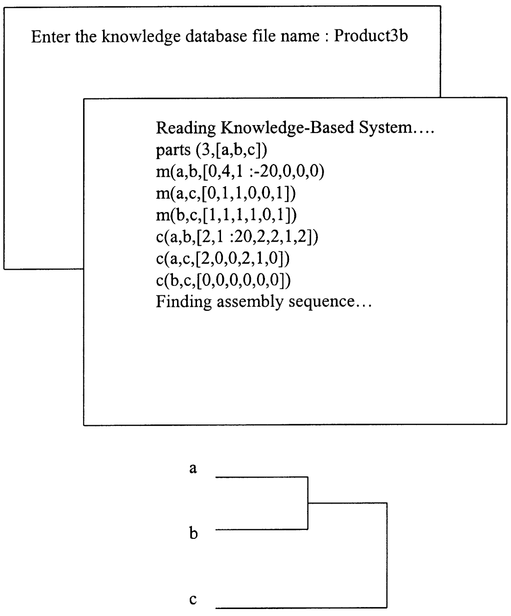
Other Automated Assembly Applications

The object-oriented techniques and automated methods eliminate the traditional steps of off-line programming. The assembly architecture, designed through artificial intelligence, solves the use of advanced curve and surface definition features in the relationship between the shapes to be manipulated.

For many years, some predefined tasks in the robotic field have been solved using sensor-based reasoning and/or the trajectory-generation algorithm [9, 12, 33]. These procedures use the constraints of the system to implement operation sequences for achieving specified goals. The basis for establishing a set of design rules for robotic and automated assembly yielded some more generalized design guidelines in 1986 [11]. In 1988, Sanderson et al. [26] introduced a method for the task planning of robotic manipulation in space application.

The major driving applications are part handling and product design. Effective and efficient utilization of assembly techniques requires that the robot task sequence be compatible with the design of components. Then, experimentation was carried out on an automated assembly of mechanical parts in a high mix product environment. The concept of replacing a certain standard simulation technique to provide motion descriptions and task parameters has been carried out by the automated assembly. It may be viewed as the tool of choice for a growing range of applications. The knowledge state of the manipulation motions and operations are defined and validated before accessing the real robot. 3-D modeling has also streamlined the tooling stage. The system is incorporating embedded task planning, with accompanying control synthesis and past architecture, to support goal-directed activities in an uncertain environment. Surface modeling has been defined and produced within tolerances as small as tenths of millimeters in order to be installed on production assemblies and still meet design requirements.

When provided with the assembly tree, the robot planner is capable of considering the objects in a given coordinate frame to deduce which features of a component mate with which others. Fig. 8.12 presents the assembly tree structure, which is handled by a robot planner to perform the specific operations related to Fig. 8.11. Fig. 8.13 shows the capability of the system to handle various levels of component complexities. According to the assembly theory, there are 6! = 720 possible solutions excluding constraints for the six components.



```
Enter the knowledge database file name : Product3b
```

```
Reading Knowledge-Based System....
parts (3,[a,b,c])
m(a,b,[0,4,1 :-20,0,0,0)
m(a,c,[0,1,1,0,0,1])
m(b,c,[1,1,1,1,0,1])
c(a,b,[2,1 :20,2,2,1,2])
c(a,c,[2,0,0,2,1,0])
c(b,c,[0,0,0,0,0,0])
Finding assembly sequence...
```

FIGURE 8.12 Robotic Assembly Sequence of the Example in Fig. 8.11.

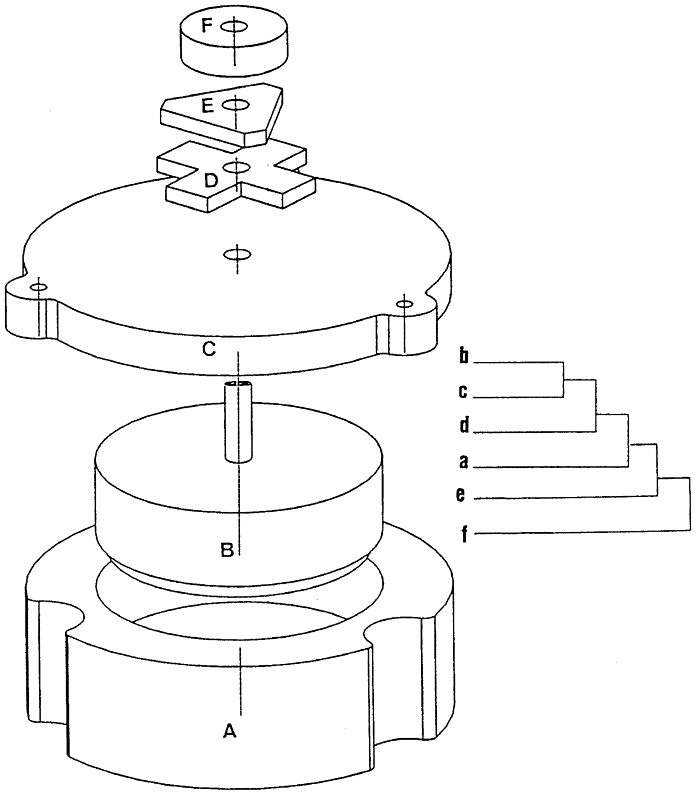


FIGURE 8.13 Equipment Layout with its Robotic Assembly Sequence.

8.5 Conclusion

This chapter discussed the object-oriented techniques and automated methods currently utilized for robotic assembly in manufacturing systems. More precisely, it addressed the problem of knowledge representation in intelligent systems dedicated to assembly planning and robotic assembly. Many real assembly-planning systems are built on the paradigm of artificial intelligence and are implemented as knowledge-based systems. Some of them aim at generating mechanical assembly sequences, which satisfy a set of assembly requirements and may be assembled by humans or robots. All the feasible assembly sequences can be generated as a tree sequence structure and then evaluated, the most appropriate one being chosen to carry out the equipment layout.

For managing geometrical data, topological and abstraction, object-oriented modeling appears to be a suitable approach. According to this course of action, data are stored as examples of abstract data types and all conceptual entities are objects. A class inherits all instances from its superclass. In this context, the question of robotic assembly planning has been discussed as a problem of generating assembly sequences derived from a model-based object recognition. Thus, the output is a list of feasible assembly sequences for use by offline robotic programming to determine the manipulator path generation. A predefined path generation can be very helpful for applications in the fields of telerobotics, machining operations, and automated assembly systems.

Robotic assembly is a difficult process. In fact, numerous transformations must be performed to capture the object features through the set of coordinate frames for each surface. Such a task can become a time-consuming iterative motion. Finally, there is a need to minimize the kinematic computation by providing the robot with knowledge of sequence generation of the components to be assembled.

References

1. Angermuller, G. and Hardeck, W. (1987), CAD Integrated Planning for Flexible Manufacturing Systems with Assembly Tasks, *IEEE CAD Journal,* pp. 1822–1826.
2. Camasinka-Matos, L.M., Seabra Lopes, L., and Barata, J. (1994), Execution Monitoring in Assembly with Learning Capabilities, *IEEE International Conference on Robotics and Automation,* pp. 272–280.
3. Cha, J., Rao, M., Zhou, Z., and Guo, W. (1991), New Progress on Integrated Environment for Intelligent Manufacturing Automation, *Proceedings of the 6th IEEE International Symposium on Intelligent Control,* Arlington, VA.
4. Chakrabasty, S. and Wolter, J. (1994), A Hierarchical Approach to Assembly Planning, *IEEE International Conference on Robotics and Automation,* Vol.1–4, pp. 258–263.
5. Chang, K.H. and Wee, W.G. (1993), A Knowledge-Based Mechanical Assembly Planning, in *Expert Systems in Engineering Applications,* S.G. Tzafestas (Ed.), pp. 291–306.
6. Christiansen, A.D. and Goldberg, K.Y. (1995), Comparing Two Algorithms for Automatic Planning by Robots in Stochastic Environments, *Robotica*, Vol. 13, No. 6, pp. 565–573.
7. Conradson, S., Weinstein, M., Wilker, J.S., and Yencho, S.A. (1987), Automated Material Handling, (Automated Assembly and Product Design as a System), *IEEE Proceedings of the 8th Int. Conf. on Assembly Automation,* pp. 67–68.
8. Ferland, M., O'Shea, J., and Rabemanantsoa, M. (1993), Robotic Interface for CAD/CAM Integration. *Proceedings of Dept. of National Defense on Workshop on Advanced Tech.* in *Knowledge-Based Systems and Robotics,* Ottawa, ON, Canada.
9. Homem de Mello, L.S. and Sanderson, A.C. (1989), A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences, *IEEE Journal of Int. Conf. on Robotics and Automation,* CH2750-8, pp. 56–61.
10. Hurt, J. (1989), A Taxonomy of CAD/CAE Systems, *Manufacturing Review,* Vol. 2, No. 3, pp. 170–178.
11. Jackson, A.J. and McMaster, R.S. (1986), Product Design for Robotic and Automated Assembly, *Proc. of IEEE International Conference on Robotics and Automation,* Vol. 2.
12. Jarvis, R.A. (1988), Configuration Space Collision-Free Path Planning for Robotic Manipulators, *Robots in Australia's Future Conference,* ARA, pp. 193–204.
13. Kamrani, A.K., Shashikumar, S., and Patel, S. (1995), An Intelligent Knowledge-Based System for Robotic Cell Design, *Computers & Industrial Engineering,* Vol. 29, pp. 1–4.
14. Kokar, M., Anderson, C., Dean, T., Valavanis, K., and Zadrozny, W. (1990) Knowledge Representations for Learning Control, *Proceedings of the 5th IEEE International Symposium on Intelligent Control,* Philadelphia, PA, USA.
15. Lee, Y.C. and Fu, K.S. (1983), A CSG Based DBMS for CAD/CAM and its Supporting Query Language, *IEEE CAD Journal,* pp. 123–128.
16. Marque-Pacheu, G., Gallausiaux, J.M., and Jormier, G. (1984), Interfacing Prolog and Relational DBMS, in *New Applications of Databases,* E. Gelenbe (Ed.), Academic Press.
17. Massey, L.L., Udoka, S.J., and Ram, B. (1995), Robotic System Design: a Hierarchical Simulation-based Approach, *Computers & Industrial Engineering,* Sept., No. 29, pp.1–4.
18. Nnaji, B.O. (1988), A Framework for CAD-Based Geometric Reasoning for Robot Assembly Language, *Int. Journal Proc. Res.,* Vol. 26, no. 5, pp. 735–764.
19. Pfeiffer, F. and Johanni, R. (1987), A Concept for Manipulator Trajectory Planning, *IEEE Journal of Robotics and Automation,* RA-3, 3, pp. 115–123.
20. Rabemanantsoa, M. and Pierre, S. (1993), A Knowledge-Based Approach for Achieving Assembly Tasks, *Proceedings of the 14th Canadian Congress of Applied Mechanics* CANCAM 93, Kingston, ON, Canada, Vol.1, pp. 51–52.
21. Rabemanantsoa, M. and Pierre, S. (1993), An Integrated Knowledge-Based System for Flexible Assembly Process Manufacturing, *2nd. Int. Conf. on Computer Integrated Manufacturing,* Singapore, pp. 789–798.

22. Rabemanantsoa, M. and Pierre, S. (1993), A Knowledge-Based System for Assembly Process-Planning, *IEEE SESS 93 Int. Conf. on Artificial Intelligence,* Brighton, England, pp. 267–272.
23. Rabemanantsoa, M. and Pierre, S. (1993), A Knowledge-Based Approach for Robot Assembly Planner, *IEEE Proceedings of Canadian Conf. on Electrical and Computer Engineering,* Vancouver, BC, Canada, pp. 829–832.
24. Rabemanantsoa, M. and Pierre, S. (1996), Robotic Assembly for Computer-Integrated Manufacturing, *International Journal of Robotics and Automation,* Vol. 11, No. 3, pp. 132–140.
25. Rabemanantsoa, M. and Pierre, S. (1996), An Artificial Intelligence Approach for Generating Assembly Sequences in CAD/CAM, *Artificial Intelligence in Engineering,* Vol. 10, No. 2, pp. 97–107.
26. Sanderson, A.C., Peshkin, M.A., and Homem de Mello, L.S. (1988), Task Planning for Robotic Manipulation in Space Applications, *IEEE Trans. on Aerospace and Electronic Systems,* Vol. 24, no. 5, pp. 619–628.
27. Shah, J. and Bhatnagar, S. (1989), Group Technology Classification from Feature-Based Geometric Models, *Manufacturing Review,* Vol. 2, pp. 204–213.
28. Shah, J.J. and Rogers, M.T. (1988), Functional Requirements and Conceptual Design of the Feature-Based Modeling System, *CAD Journal,* Vol. 5, pp. 9–15.
29. Swift, K.G. (1987), Knowledge-Based Design for Manufacture, Prentice-Hall, London, England.
30. Tsao, J. and Wolter, J. (1993), Assembly Planning with Intermediate States, *IEEE International Conference on Robotics and Automation,* Vol. 1–3, pp. 71–78.
31. Thompson, D.R. and Ray, A. (1987), A Hierarchically Structured Knowledge-Based System for Welding Automation and Control, *Proceedings IEEE International Symposium on Intelligent Control,* Philadelphia, PA, USA.
32. Valavanis, K.P. and Tzafestas, S.G. (1993), Knowledge-Based (Expert) Systems for Intelligent Control Applications, in *Expert Systems in Engineering Applications,* S.G. Tzafestas (Ed.), pp. 259–268.
33. Woodbury, R.F. and Oppenheim, I.J. (1988), An Approach to Geometric Reasoning in Robotics, *IEEE Trans. on Aerospace and Electronic Systems,* Vol. 24, no. 5, pp. 630–645.