An Object-Oriented Approach to Generate Mechanical Assembly Sequences

by

Wen Hu

A thesis

Presented to the University of Manitoba

in Partial fulfilment of the

requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

Winnipeg, Manitoba, Canada, 1997

©Wen Hu

National Library of Canada

Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services

Acquisitions et services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-23346-4

Canada

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES
*****
COPYRIGHT PERMISSION PAGE

AN OBJECT-ORIENTED APPROACH TO GENERATE MECHANICAL ASSEMBLY SEQUENCES

BY

WEN HU

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree

of

MASTER OF SCIENCE

Wen Hu 1997 (c)

Permission has been granted to the Library of The University of Manitoba to lend or sell copies of this thesis/practicum, to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film, and to Dissertations Abstracts International to publish an abstract of this thesis/practicum.

The author reserves other publication rights, and neither this thesis/practicum nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

Abstract

A mechanical assembly is a composition of parts interconnected to form a stable unit.The mechanical assembly sequence problem is to find a feasible, cost-effective sequence of tasks to perform the assembly. This problem has recently been recognised as significant by the manufacturing industries, since this plan will help mechanical designers to analyse the assembly tasks off-line and justify the cost involved in the process. Homen de Mello and Sanderson proposed the *AND/OR* graph notation to represent all feasible assembly sequences for a given assembly in one graph and also provide an algorithm to generate the *AND/OR* graph.

Recently, object-oriented approach to software design becomes popular in the computer science community, mainly because of its advantages over the functional approach towards software reuse and maintenance. The object-oriented approach asserts that enhancement and adaptation of the software to the environment are easier to do, compared to the functional approach. In thesis, we have used the object-oriented approach to redesign the AND/OR graph generation algorithm for assembly sequence generation. This approach inherits the advantages of the object-oriented paradigm.

The main contributions of this thesis are the following:

- An object-oriented version of the assembly sequence algorithm to generate *AND/OR* graph notation is given.
- This version eliminates some of the redundant steps in the functional algorithm.
- The object-oriented design described in this thesis is easy-to-understand, easy-to-enhance and inherits the advantages of the object-oriented paradigm.
- A formal model of the object-oriented design is also included in the thesis.

Acknowledgements

First of all I would like to give my deepest gratitude to my supervisor, Dr. Kasi Periyasamy for his guidance, advice and encouragement. I wouldn't be able to finalize the whole work within one month without his help. I feel very lucky to have him as the supervisor.

My thanks go to my parents, my brothers' family and my sister for their support and concerns. I also need to thank my daughter for her very good performance during I am away. Finally my thanks go to my husband for his emotional support all the time.

List of Figures

Figure 1: An Assembly with five parts -(9)

Figure 2: The relational graph of the assembly shown in Figure 1 - - - - - (10)

Figure 3: The graph of connections of the assembly shown in Figure 1 - - - (10)

Figure 4: AND/OR graph for the assembly shown in Figure 1 - - - - - - - (20)

Figure 5: The Object Model - (36)

Figure 6: data-consistency - Phase 1 - (38)

Figure 7: data-consistency - Phase 2 - (39)

Figure 8: data-consistency - Phase 3 - (40)

Figure 9: data-consistency - Phase 4 - (41)

Figure 10: Gen-feasible-decompositions() - - - - - - - - - - - - - - - - - - (43)

Figure 11: isConnected() - (45)

Table of Contents

Chapter 1: Introduction -(1)

Chapter 2: AND/OR Graph Generation - Functional Approach - - - - - - (6)

2.1: Input to AND/OR Graph Method - - - - - - - - - - - - - - - - - -(6)

2.2: AND/OR Graph Generation - - - - - - - - - - - - - - - - - - - (14)

2.3: Generation of Assembly Sequences - - - - - - - - - - - - - - (15)

2.3.1: Overview - (15)

2.3.2: Algorithm - (16)

2.4: Formal Description of AND/OR Graph Functional Model - - - (21)

Chapter 3: AND/OR Graph Generation - Object - Oriented Approach - - - (22)

3.1: Overview - (22)

3.2: Class Definition - (23)

3.3: Interactions among the objects - - - - - - - - - - - - - - - - - (37)

3.3.1: Generate the Assembly - (37)

3.3.2: Generate Feasible Decompositions - - - - - - - - - - - - - (42)

3.3.3: Generate AND/OR Graph - - - - - - - - - - - - - - - - - - (46)

3.3.4: Example - (47)

3.4: Formal Description of the Object-Oriented approach - - - - - - (48)

Chapter 4: Comparison - (54)

4.1: The equivalence of the two approaches - - - - - - - - - - - - - (54)

4.2: The advantages of our approach over Homen De Mello and Sanderson's approach - (56)

4.3: Limitation of both the approaches - - - - - - - - - - - - - - - (57)

Chapter 5: Conclusion and Future Work - - - - - - - - - - - - - - - - - - (58)

5.1: Conclusion - (58)

5.2: Future Work - (59)

Appendix A: Formal Notations - (60)

Appendix B: AND/OR graph for the example assembly - - - - - - - - - (61)

Appendix C: Object Model Notation - (63)

Bibliography - (64)

Chapter 1

Introduction

A mechanical assembly is a composition of parts interconnected to form a stable unit[10]. The process of assembling a product consists of several tasks; each task describes how to join two or more subassemblies to form a larger subassembly. This process starts with individual parts and ends with all parts joined properly to form the final product. The mechanical assembly sequence problem is to find a feasible, cost-effective sequence of tasks to perform the assembly. For the purposes of this thesis, each task in the assembly process accepts at most two subassemblies as input, and a part by itself becomes a subassembly.

The problem of generating a correct and feasible assembly sequence has recently been recognised as significant by the manufacturing industries, since this plan will help mechanical designers to analyse the assembly tasks off-line and justify the cost involved in the process. For example, the difficulty of assembly steps, the need for fixtures, the potential damage that could occur to the parts during assembly, the occurrence of tool changes and thus the cost of the assembly are all affected by the choice of the assembly sequence.

Traditionally, the sequence of assembly tasks is decided by a human expert[9]. For example, in a manufacturing plant the sequence is planned by an experienced industrial engineer and in repair work it is planned by a maintenance personnel. However, humans tend

to make mistakes, particularly with larger assemblies. There will always exist the possibility that a good assembly sequence has been overlooked. The more complex the product is, the more possibility that a good assembly sequence will be overlooked. Hence, it is evident that a systematic and computerized mechanism is needed to plan the assembly sequence. Moreover, automation of this process is feasible with the computerized mechanism in place; in addition, this process can be linked to other tasks in the industry.

In this thesis, we focus on algorithms to generate all the feasible assembly sequences for a given mechanical product so that a mechanical engineer can pick up a suitable solution among the alternatives. There is a considerable difference between assembling a mechanical product and assembling electronic components on a printed circuit board. This thesis is devoted to the former category and hence is not related to the electronic components assembly.

A lot of research has been done in solving assembly sequence problem. Several methods have been proposed to generate the assembly sequences from the geometric descriptions of the final product and the components. A brief introduction to some of these methods is given below:

Homen de Mello and Sanderson proposed an *AND/OR* graph notation to represent all feasible assembly sequences in one graph[9, 10]. Each node in the graph represents a subassembly showing all of its possible decompositions, thus showing the OR component. Each decomposition shows the ANDing of two subassemblies. The *AND/OR* graph in this case is generated using a disassembly process which uses the representation of the final product as its input. In this approach, the problem of disassembling a subassembly is decomposed into two distinct subproblems, each being to disassemble one subassembly. The key point in this process is to find out whether the decomposed subassemblies can be reassembled by a reversible disassembly process. If so, that particular step of disassembly is included in the set of feasible decompositions. The process continues until the whole

assembly is disassembled into individual components. This approach lends itself to an *AND/OR* graph representation of assembly sequence[9]. The whole process is automatable. The *AND/OR* approach is an industrially successful method because of its natural decomposition; i.e. any design approach uses *AND/OR* decomposition intuitively.

De Fazio and Whitney proposed a method called *precedence relation graph*[6]. In this approach, each assembly task is associated with another task having some precedence over the other. The partial ordering among the precedence of the assembly tasks gives rise to a correct sequence. This approach accepts the information on parts and the "user_defined" relations between parts called "*liaisons*"; it then enumerates these *liaisons*, asking two questions: 1) what liaisons must be done prior to doing this liaison? 2) what liaisons must be left to be done after doing this liaison? The whole assembly sequence is then computed by applying the answers collected from the user for the questionnaire.

Peral Pu proposed a method called *cased-based* search techniques[19]. It solves the assembly sequence problem by retrieving a solution from its *case* library which is derived from solving similar problems in the past and then adapting the solution to the new problem. Each *case* in the *case-base* is a solved assembly problem.

In the first two cases, the establishment of a feasible assembly sequence as well as its correctness are discussed by the respective groups. In addition, both approaches give all feasible assembly sequences for a given set of parts and its final assembly. But the approach proposed by De Fazio and Whitney accepts very little information from the user compared to the *AND/OR* graph approach. This leads to considerable problems in automating the *precedence relation* approach. Another problem with the *precedence relation* approach is that its inexplicit representation makes it harder to understand and use.

The *Case-based* method mainly depends on the case library. If a case does not exist in the library, the system creates a new case and updates its library. Consequently, automation of

this approach is relatively tedious. Moreover, the size of the library determines the complexity of the algorithms used and the efficiency of the application.

Some other approaches have also been reported in the literature[13, 24].All the methods proposed so far are functional since they concentrate on the tasks to be performed[9,6, 20]. Recently, object-oriented approach to software design becomes popular in the computer science community, mainly because of its advantages over the functional approach towards software reuse and maintenance[2, 20]. The former asserts that enhancement and adaptation of the software to the environment are easier to do, compared to the functional approach. Therefore, it is decided by the author to adapt an object-oriented approach in this thesis to redesign the assembly sequence generation algorithms to gain advantages such as reuse and enhancement.

Since our purpose is to show the advantages of the object-oriented approach, we do not plan to invent a new algorithm for assembly sequence planning. Rather, we redesign Homen de Mello and Sanderson's *AND/OR* graph generation algorithm because of its several advantages over other functional approaches reported in the literature. Informally, we justify that the object-oriented model captures all properties of the functional model and show how the object-oriented model can be enhanced to include additional information such as material and functionality of the components in order to improve the assembly sequence generation algorithm.

In summary, the contributions of this thesis are given below:

- An object-oriented version of the assembly sequence algorithm using *AND/OR* graph notation is given in this thesis.

- This version eliminates some of the redundant steps in the functional algorithm at the cost of introducing redundant data information. Details of this claim are given in Chapter 5.

- The object-oriented design described in this thesis is easy-to-understand, easy-to-enhance and inherits the advantages of the object-oriented paradigm.

- A formal model of the object-oriented design is also included in this thesis.

The organization of the thesis is as follows: Chapter 2 briefly describes the functional approach given by Homen de Mello and Sanderson. For more details, readers are referred to [9]. Chapter 3 describes in detail the object-oriented design of the *AND/OR* generation algorithm and its formal representation. The comparison between the two approaches are given in Chapter 4. The thesis concludes in Chapter 5 with comments on future work.

Chapter 2

AND/OR Graph Generation - Functional Approach

In this Chapter, we briefly describe the function approach to generate AND/OR graph. This work was done by Homem de Mello and Sanderson; more details about this approach can be found in [9]. We choose the method based on AND/OR graph notation in this thesis because (i) it is an industrially successful method; (ii) compared to the other methods, the AND/OR graph based method is easier to automate; and (iii) the method was an intuitive notion of decomposition based on disassembly approach; such an approach is easier to understand and implement.

Hereafter, we use the term "AND/OR graph method" in this Chapter to refer to the functional approach for generating assembly sequences.

2.1 Input to AND/OR Graph Method

The AND/OR graph method starts with a relational graph of an assembly which describes the parts and their interconnections making up the assembly.

Formally, the relational graph of an assembly is a quintuple <P, C, A, R, *a-function*> where

- P is a set of symbols, each of which uniquely identifies a part in the assembly.

- C is a set of symbols, each of which uniquely identifies a contact between exactly two parts in the assembly.

- A is a set of symbols, each of which uniquely identifies an attachment that acts on a contact. Typically, an attachment describes the physical media of the contact, such as glue, screw, etc.

- R is a set of symbols, each of which uniquely describes a relationship between pairs of elements among parts, contacts and attachments. The purpose of defining a relationship is to identify the role of entities during an assembly task. For example, during a screw assembly, one part serves as an agent being driven and the others part serves as the target.

- *a-function* is a set of attribute functions, each of which uniquely associates the entities or relationships to their characteristics. For example, an *a-function* may return the shape of a part, the location of a part or the type of a contact. An *a-function* can be modified to include any additional information that is necessary to generate assembly sequences.

The following assumptions are made with respect to the examples used in this Chapter and in the rest of the thesis.

- The types of contacts considered are: *planar-to-planar, cylindrical shaft-to-cylindrical hole, polyhedral shaft-to-polyhedral hole,* and *threaded cylindrical shaft-to-threaded cylindrical hole.*

- The type of attachment considered are *glue, clip, pressure fit attachment,* and *screw attachment.*

- The types of relationships included are *part-contact relationship, target-attachment relationship* and *agent-attachment relationship. Part-contact relationship* is the relationship between the part and the contact. Every contact must have exactly two *part-contact relationships* and every part must have at least one *part-contact relationship. Target-attachment/agent-attachment relationship* is the relationship between the attachment and its target/agent. Every attachment must have at least one *target-attachment* and at least one *agent-attachment relationship.*

- These classifications can be expanded/modified without affecting the algorithms or the method.

To illustrate, consider the example of an assembly whose parts are given in Figure 1; the corresponding relational graph of the assembly given in Figure 2. This graph shows all elements of P, C, A and R from the quintuple. A simplified view of the relational graph showing only the parts and contacts given in Figure 3. Even though this view is redundant, we have included it here because all the papers by Homen de Mello and Sanderson include both versions of the relational graph. In their terminology, the simplified version of the relational graph is called "graph of connections".

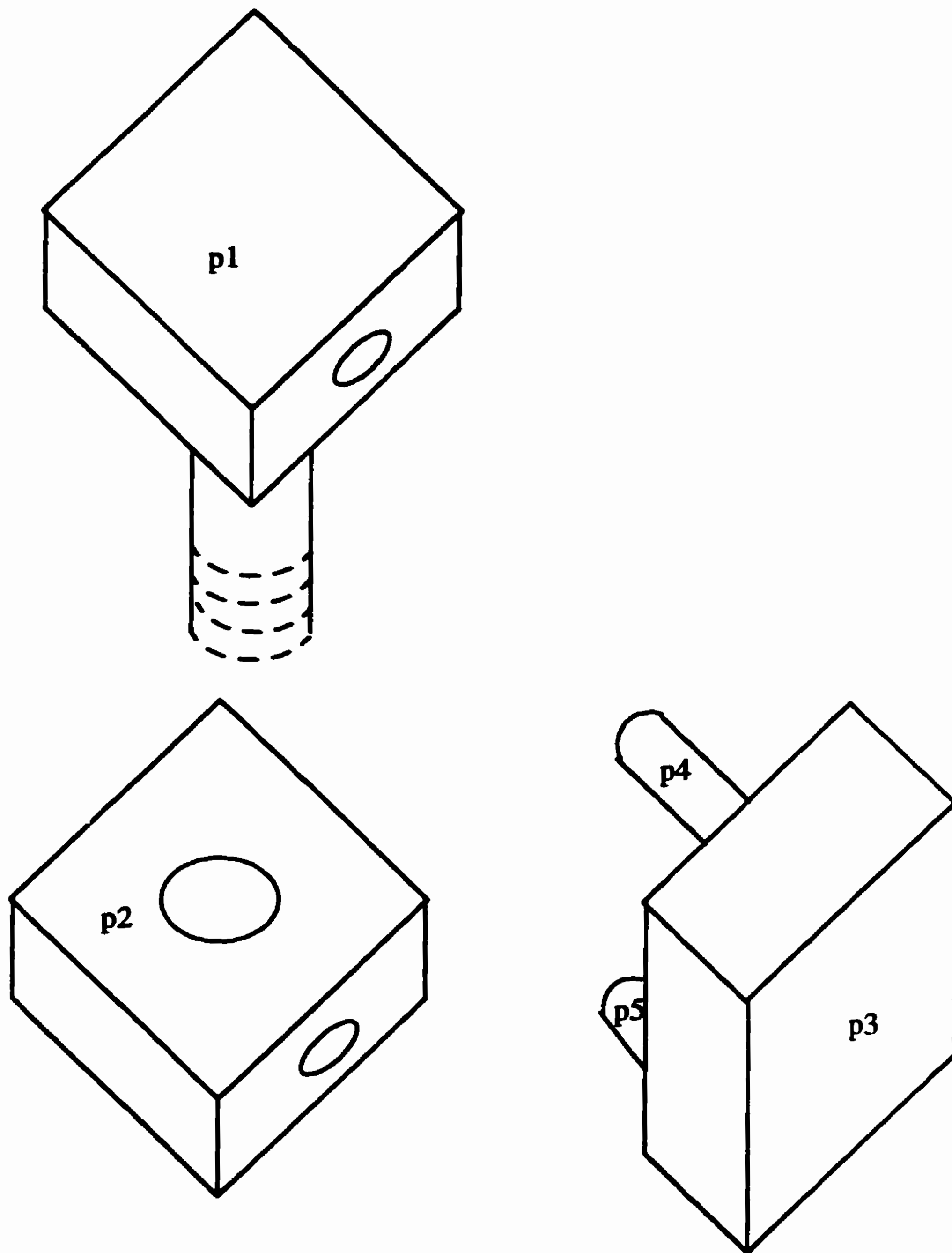


Figure 1 An assembly with five parts

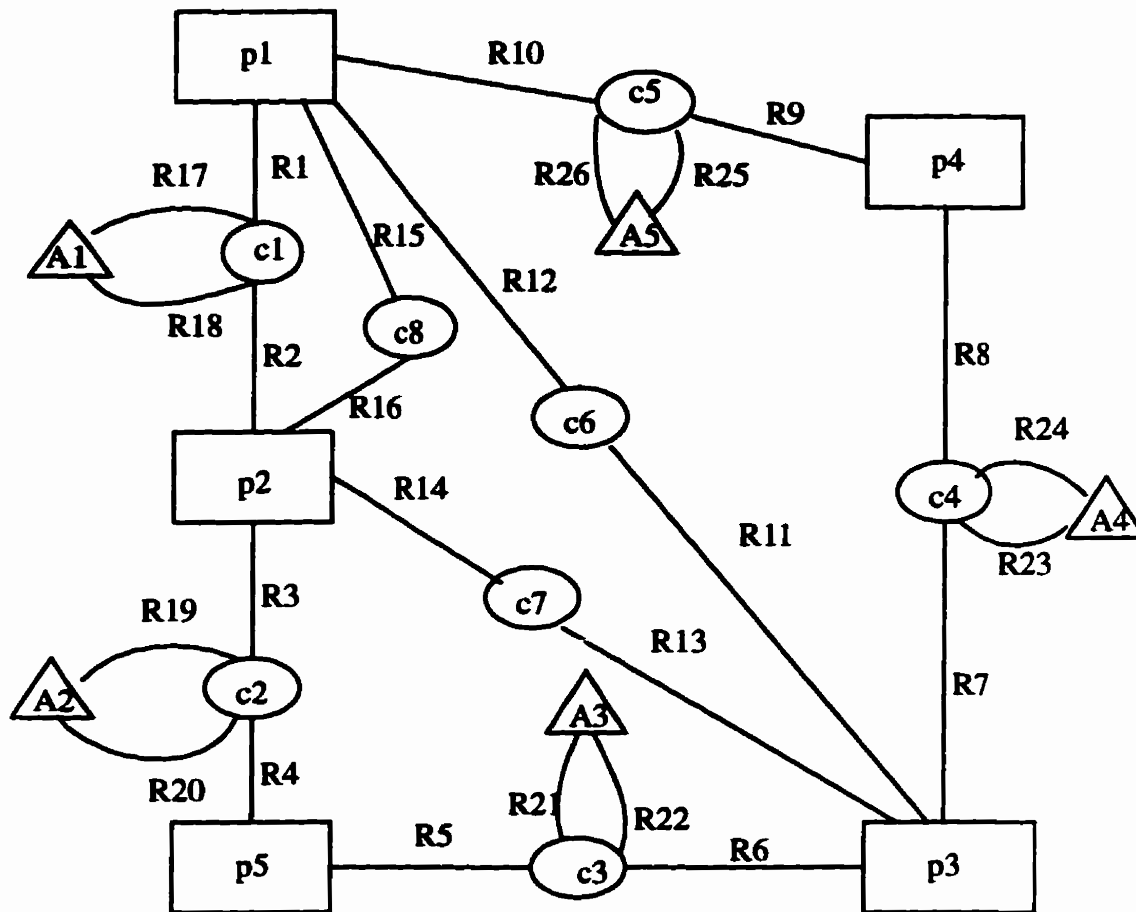


Figure. 2 The relational graph of the assembly shown in Figure 1

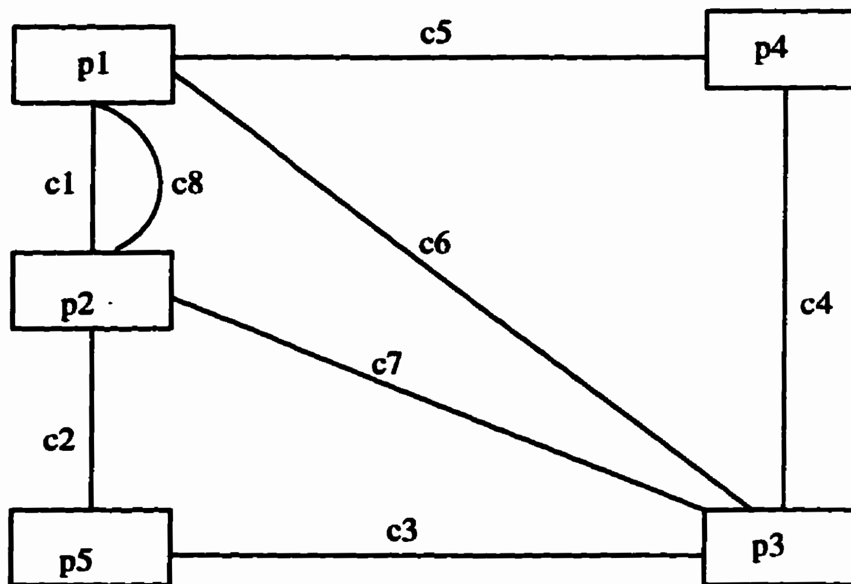


Figure. 3 The graph of connections of the assembly shown in Figure 1

The elements of the quintuple are:

P = {P1, P2, P3, P4, P5}

C = {C1, C2, C3, C4, C5, C6, C7, C8}

A = {A1, A2, A3, A4, A5}

R = {R1, R2, R3, ..., R26}

The *a-function* for this assembly are categorized as follows:

1. The functions that associate a part to a description of its shape.
2. The functions that associate a contact to its type.
3. The functions that associate a planar contact to the coordinates, with respect to the assembly's global frame of references, of a vector normal to the planar contact.
4. The functions that associate a planar contact to its Forward part, which is the part that the normal to the plane of contact pointing to the exterior of the part.
5. The functions that associate a planar contact to its Back part, which is the part that the normal to the plane of contact pointing to the interior of the part.
6. The functions that associate a cylindrical, a slot, or a thread_ cylindrical contact to its coordinates, with respect to the assembly's global frame of reference, of the line of the axis of both the hole and the shaft.
7. The functions that associate an attachment to its type.
8. The functions that associate a relationship to its type.
9. The functions that associate a part_contact relationship to its part.
10. The functions that associate a part_contact relationship to its contact.
11. The functions that associate an attachment or a contact in an agent_attachement relationship.

12. The functions that associate an attachment contact in a target_attachment relationship.
13. The functions that associate an agent_attachment relationship to its agent.
14. The functions that associate an target-attachment relationship to its target.

The *a-functions* are shown in Table 1 - Table 4.

TABLE 1.

part	p1	p2	p3	p4	p5	a-function in category
shape	plannar_cylindrical_combination	planner	planner_cylindricalhole_combination	cylindrical	cylindrical	1

TABLE 2.

contact	C1	C2	C3	C4	C5	C6	C7	C8	a-function in category
type	threaded_cylindrical	cylindrical	cylindrical	cylindrical	cylindrical	planar	planar	planar	2
Normal						(1 0 0)	(1 0 0)	(0 0 1)	3
Forward						P3	P2		4
Back						P1	P2		5
Coordinate	(0 0 2) (0 0 1)	(2 0 0) (0 0 0)	(2 0 0) (3 0 0)	(2 1 0) (3 1 0)	(2 1 0) (0 1 0)				6
target-attachment relationship	R18	R19	R22	R23	R26				12
agent-attachment relationship	R17	R20	R21	R24	R25				part of item 11

TABLE 3.

attachment	A1	A2	A3	A4	A5	a-function in category
type	screw	pressure	pressure	pressure	pressure	7

TABLE 4.

Relationship	type	related-part	related-contact	related-attachment	agent	target
R1	part-contact	P1	C1			
R2	part-contact	P2	C1			
R3	part-contact	P2	C2			
R4	part-contact	P5	C2			
R5	part-contact	P5	C3			
R6	part-contact	P3	C3			
R7	part-contact	P3	C4			
R8	part-contact	P4	C4			
R9	part-contact	P4	C5			
R10	part-contact	P1	C5			
R11	part-contact	P3	C6			
R12	part-contact	P1	C5			
R13	part-contact	P3	C7			
R14	part-contact	P2	C7			
R15	part-contact	P1	C8			
R16	part-contact	P2	C8			
R17	agent-attach-ment		C1	A1	P1	
R18	target-attach-ment		C1	A1		P2
R19	target-attach ment		C2	A2		P2
R20	agent-attach-ment		C2	A2	P5	
R21	agent-attach-ment		C3	A3	P5	
R22	target-attach-ment		C3	A3		P3
R23	target-attach-ment		C4	A4		P3

Relationship	type	related-part	related-contact	related-attachment	agent	target
R24	agent-attach-ment		C4	A4	P4	
R25	agent-attach-ment		C5	A5	P5	
R26	target-attach-ment		C5	A5		P1
a-function in Category	8	9	10	11	13	14

The relational graph provides a data structure that maintains contact geometry and connectivity information at one level of representation and completed part geometry at a second level[9]. As observed from the set of data, most of the *a-functions* are meant to describe an attribute or property of a particular entity. In our model, we have simplified these *a-functions*; details on these simplifications are described in the next chapter.

2.2 AND/OR Graph Generation

An AND/OR graph consists of a set nodes corresponding to subassemblies created during the assembly process, and a set of hyperarcs emanating from each subassembly. Each hyperarc shows two subassemblies which are joined to make up the subassembly under consideration (which represents AND). The set of hyperarcs emanating from a subassembly node shows the possible decompositions of a given subassembly (which represents OR). During actual assembly, only one of the possible decompositions is considered. Selection of a particular decomposition depends on serval factors such as cost, tool support and other organizational concerns. Thus, an AND/OR graph represents all possible assembly decompositions and sequences of assembly task. Any path from the leaves (the separated parts) to the root (the final assembly) of the AND/OR graph is a valid assembly.

2.3 Generation of the Assembly Sequences

In this section, we describe in detail the algorithms of the AND/OR graph method.

2.3.1 Overview

Homem de Mello and Sanderson transformed the problem of generating assembly sequences to the problem of generating disassembly sequences. The following assumptions have been made during the generation of the AND/OR graph:

- Each disassembly task is the reverse of a feasible assembly task.
- Exactly two parts or subassemblies are joined at each time.
- All the contacts between two parts or subassemblies are established when these two parts or subassemblies are joined.
- All the contacts within the subassemblies still remain when an assembly is disassembled into two subassemblies.

A decomposition approach was used to solve the disassembly problem. Every decomposition corresponds to a disassembly task. Every reversible disassembly task is then a valid assembly task which can be tabulated. Each disassembly step creates one decomposition which is a pair of subassemblies. The decomposition is checked for feasibility(i.e. confirming the possibility of a reverse assembly) based on two criteria: task-feasibility and subassembly-stability criteria. The task-feasibility predicate is true if there exists a mechanical process to join the two subassemblies. The subassembly-stability predicate is true if the parts in each subassembly maintain their relative position and do not break contact spontaneously. The process continues until the whole assembly is disassembled into

the components. This approach lends itself to an AND/OR graph representation of assembly sequence.

2.3.2 Algorithms

The basic idea of the algorithm is to enumerate all possible decompositions first and then retaining only those that pass the check on feasibility. The AND/OR graph generation is then to develop the graph to show all feasible sequences in one graph.

Two procedures were given by Homem de Mello and Sanderson: *GET_FEASIBLE _DECOMPOSTION* and *GENERATE_AND_OR_GRAPH*. Before describing these two procedures, let us start the with the concept of *cut_set*.

CUT-SET

A *cut-set* describes the possibility of a decomposition in a given assembly. For example, in a 5-parts assembly, a decomposition can be achieved by breaking one or more contacts resulting in one 2-part and one 3-part subassemblies. The set of contacts must be broken in order to generate the decomposition is called a *cut-set*

***Algorithm: GET_FEASIBLE_DECOMPOSTION*:**

- This algorithm accepts the relational graph as input.

- It first generates the graph of connection from the relational graph by calling a procedure GET_GRAPH_OF_CONNECTION and then computes the *cut_sets* from this graph. Remember that *cut-sets* denotes the set of all *cut-sets* for the graph.

- For every *cut_set* in the *cut_sets*, a decomposition is generated by calling the procedure *GET_DECOMPOSITION*.

- Each decomposition is then subject to the feasibility test. If feasible, the decomposition is included in a list called *fsl-dec*.

- The algorithm finally returns fsl-dec.

We now illustrate the algorithm using the example shown in Figure 1

Step 1: The cut_sets *CS* is generated

CS = {{C2, C3}, {C5, C6, C7, C3}, {C1, C2, C3, C8}, {C1, C4, C6, C8}, {C4, C5}, {C2, C7, C6, C4}, {C2, C6, C7, C5}, {C1, C2, C8}, {C1, C5, C6, C8}, {C3, C4, C6, C7}}

Step 2: For every cut_set in *CS*, generate its corresponding decomposition. The list of decompositions *dec* thus generated is:

{p1, p2, p3, p4} {p5}	*{p1, p2, p5}{p3, p4}*	*{p2, p5}{p1, p3, p4}*
{p2, p3, p5} {p1, p4}	*{p4} {p1, p2, p3, p5}*	*{p3, p5}{p1, p2, p4}*
{p3, p4, p5} {p1, p2}	*{p2}{p1, p3, p4, p5}*	*{p1}{p2, p3, p4, p5}*
{p3}{p1, p2, p4, p5}		

In the above list, the two sets in each decomposition indicates the parts in each subassembly.

Step 3: For each decomposition in the list *dec*, exercise the feasibility test by calling the procedure *feasible-test*. The feasibility test will use all the information about the assembly

given by the users, such as the *attachments, relationships* and *a-functions*. As an example, consider the decomposition{p5}{p1, p2, p3, p4}. This decomposition will fail during the feasibility test for the following reasons: Part p5 has contacts with p2 and p3 via the contacts c2 and c3 respectively. Parts p2 and p3, in turn, are connected through the contact c7. From the information such as the axis of alignment of the hole and shaft in p2 and p3 respectively, it is inferred that c2 and c3 must be made before c7. However, the current decomposition shows that c7 must be made before c2 and c3 which is contradictory. Therefore, this decomposition does not generate a feasible assembly task.

Homem de Mello and Sanderson have included a number of procedures in their implementation to automate the feasibility test, most of these procedures computes feasibility based on engineering calculations. In this thesis, we do not include these procedures. Instead, we display the set of information on the screen and let the user decide on the feasibility.

The *feasible_test* includes *task_feasiblites* test and *subassembly_stability* test. The automatic generation of AND-OR graph is based on the assumption that there exists a correct algorithm for computing these two tests.

Therefore, assume that we have a correct algorithm for computing *feasible_test*, the result from enumerating the decomposition list which is feasible decomposition list *fsl_dec* will be:

{p1, p2, p5} {p3, p4} *{p3, p5} {p1, p2, p4}* *{p3, p4, p5} {p1, p2}*

{p3} {p1, p2, p4, p5}

Algorithm: GENERATE_AND_OR_GRAPH

This algorithm generates the AND/OR graph from the list of feasible decompositions. Two lists are used in this procedure: ***open list*** and ***closed list*** both pointing to the relational graph of the assembly. The list ***open list*** stores the subassemblies that are not yet decomposed and *closed list* stores the one that are already decomposed. Actually, this algorithm works in conjunction with the decomposition algorithm. In summary.

- For each item in *open list*, generate all feasible decompositions by calling the procedure *GET_FEASIBLE_DECOMPOSTION.*

- Using the pointers to the relational graph by calling the procedure *GET_POINTS*, check whether each subassembly has appeared before. If so, ignore the subassembly. Otherwise, create a new pointer and insert it into the *open* list.

- Each decomposition yields one hyperarc in the AND/OR graph.

- Move the element to the *closed list.*

- Execute the loop until the *open list* is empty.

Figure 4 shows the AND-OR graph for assembly in Figure. 1

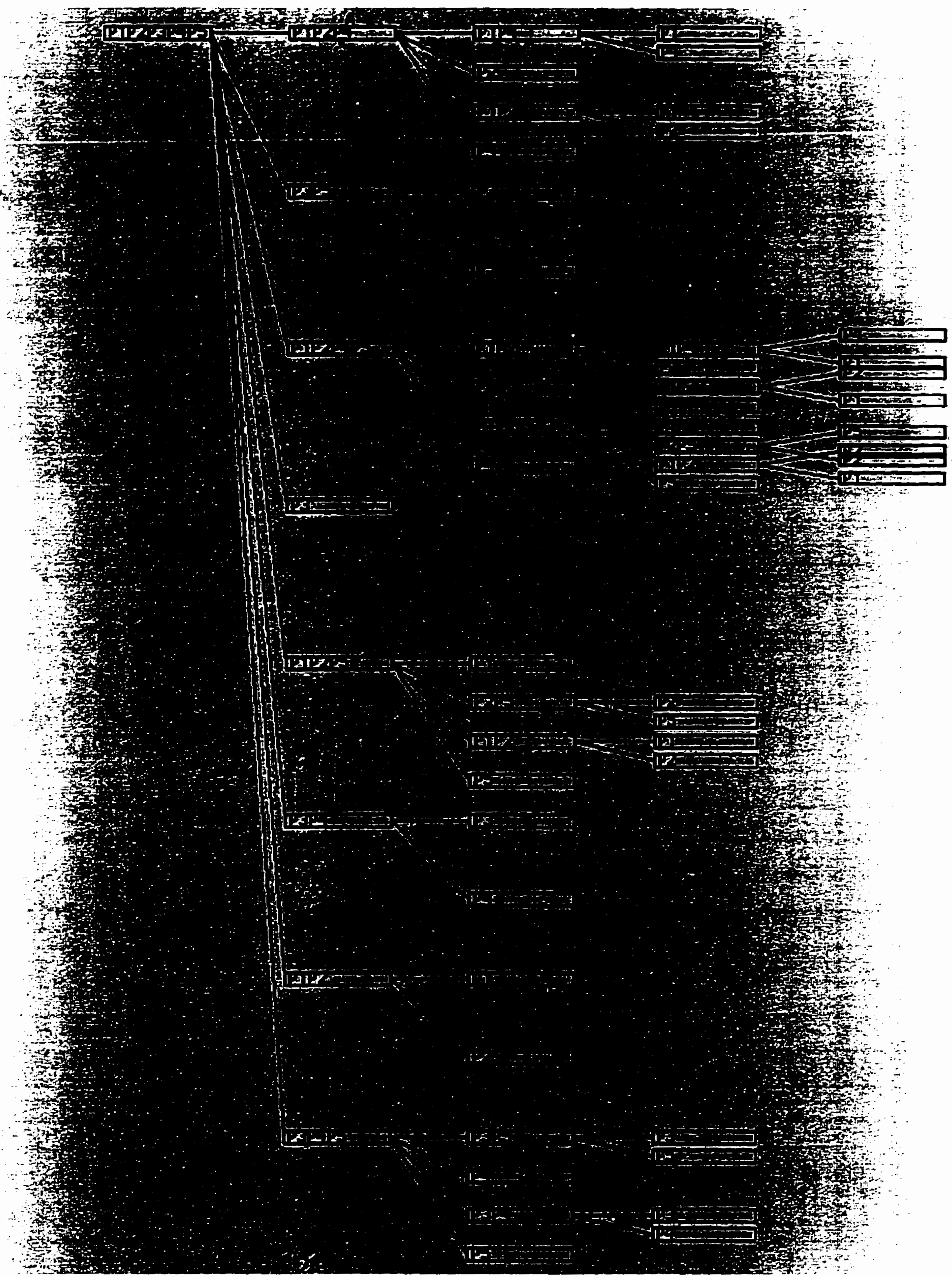


Figure 4 AND/OR graph for the assembly shown in Figure 1

2.4 Formal Description of AND/OR Graph Functional Model

As given in [9], an abstract functional model of an assembly is represented by a set of parts and a set of contacts. This model should reflect the initial graph of connections of the assembly. The assembly process is described by a sequence of assembly states, where at each assembly state, exactly two subassemblies are joined together. Initially, every part forms a subassembly. At the final stage, there is only one subassembly corresponding to the whole assembly.

A formal model of the AND/OR graph for a given set of parts $P=\{p_1, p_2, \ldots, p_N\}$ is given below: [9]

$$S_p = \{\theta \in \Pi(P) \mid sa(\theta) \wedge st(\theta)\}$$

$$D_p = \{(\theta_k, \{\theta_i, \theta_j\}) \mid \theta_i \in S_p \wedge \theta_j \in S_p \wedge \theta_k \in S_p \wedge$$

$$\theta_k = \theta_i \cup \theta_j \wedge mf(\{\theta_i, \theta_j\}) \wedge gf(\{\theta_i, \theta_j\})\}$$

Where θ_i and θ_j represent the subsets of parts from which two subassemblies are assembled; θ_k represents the subassembly which is derived from θ_i and θ_j; $\Pi(p)$ is the set of all subsets of P (power set of P); S_p represents the set of all valid and stable subassemblies which are indicated by $sa(\theta)$ and $st(\theta)$ respectively; D_p represents the set of all mechanically feasible and geometrically feasible assembly tasks which are represented by $mf\{\theta_i, \theta_j\}$ and $gf\{\theta_i, \theta_j\}$ respectively.

Chapter 3

AND/OR Graph Generation - Object-Oriented Approach

In this Chapter, we describe the AND/OR graph generation using an object-oriented approach. This chapter provides the major contribution of the thesis.

3.1 Overview

An object-oriented approach to software design is a new way of thinking about problems using models organized around real-world concepts. The fundamental construct in this approach is the object, which combines both data structure and behaviour in a single entity[9].

The object-oriented approach concentrates on designing objects and their individual behaviours. The system is represented as a collection of objects and the system tasks are performed by interactions among these objects. Generally, the designer of an object-oriented system starts by designing the real-world objects first and then adds system objects which support the realization of the real-world objects.

In object-oriented terminology, a class represents a group of objects with similar properties (attributes), common behaviours (operations), common relationships to other objects,

and common semantics[9]. By grouping objects into classes, we can abstract the problem, which is the heart of the object-oriented design.

For the AND/OR graph generation using the object-oriented approach, we start with class definitions, and then describe the interactions among these classes; an example is given to show how the AND/OR graph is generated by the object-oriented approach. Finally, we also give the formal model of an object-oriented design to generate AND/OR graphs.

We use the Object-Modeling Technique (OMT) by Rumbaugh and others to represent the design. OMT supports three views of the system: Object model, dynamic model and functional model. The object model captures the static structure of the system. The object model of the proposed design describes the structure of objects in the system - their identities, relationships to other objects and attributes, and operations in each object. The dynamic model describes the behavioural aspects of objects individually. The behaviour of an object is represented using a state transition diagram. Consequently, the dynamic model consists of a collection of state transition diagrams, one for each class. The functional model gives a transformational view of the system and is represented by data-flow diagrams. In our approach, we do not use the dynamic and functional model. Instead, we describe the algorithm using flowcharts.

3.2 Class definition

To start with, we introduce the object-oriented design model for an assembly. The object-oriented model is expected to be synonymous to the functional model described in the Chapter 2 in the following sense:

- The model should capture all the information in the relational model which is the input of AND/OR graph generation using the functional approach.

- The model should generate AND/OR graph which is the output of the functional approach.

We start our discussion by modeling individual classes. A class definition should include a structure and behaviours, one of them may be empty. The main goal in the design of a class is to see that a class definition is more or less self-descriptive; in other words, it is encapsulated and describes a part of the real-world problem.

We extract the structure and behaviour of each class definition from the entities in the functional model. The functional model describes five major entities: parts, contacts, attachments, relationships and *a-functions*. The first three can be modelled as individual classes since they describe fairly independent and distinct sets of information. A relationship describes the connectivity among parts, contacts and attachments and partly adds some information about the nature of the relationship and the role played by each of these entities (such as agent and target). Hence, a relationship can also be modelled as a class. The *a-functions* in the functional model describe the characteristics of each of these entities. Typically, each function corresponds to an attribute of the entity. Therefore, in our approach, we do not model them separately; instead, we distribute the information extracted from the *a-functions* to the various class definitions as attributes and functions within the classes.

The details of the individual class definitions follow:

1. CLASS PART

Structure:

Each part belonging to the class PART must have a unique identifier and a unique type. The identifier corresponds to the symbols *P1, P2, ..., PN* as used in the func-

tional model. The shape of a part serves as a reasonable classification of its type. Even though complex shapes are difficult to express, we do not take into account the naming of such complex shapes. A part must have a position, expressed as a vector in a global coordinate frame. Additional information such as the material of the part can be included. In addition, for encapsulation purposes, a part also maintains information about the contacts and relationships in which the part is involved. Even though it is redundant, we need to have at least pointers to the contacts and relationships for easy manipulation of relevant information during AND/OR graph generation.

Behaviour:

The only behavioural aspect of a part interesting to the current problem is the movement of the part during the assembly process. When a part moves, its position changes. A move operation could be simple such as translation or rotation, or complex such as a threaded path movement or screwing. Complex movements can be modeled as combinations of simple translations and rotations. So, we include only translation and rotation in the behavioural section of a part.

Definition:

```
class PART {
data members:
   identifier: String
   relatedContacts: set of CONTACT;
   relatedRelationships: set of RELATIONSHIP
```

shape: {planar, cylindrical, thread-cylindrical, polyhedral, complex}

position: Vector;

material: {... material name of the part...}

member functions:

void rotate (axis: DIRECTION; angle: REAL);

void translate (dir: DIRECTION; distance: REAL);

};

2 Class CONTACT

Structure:

Every contact belonging to the class CONTACT must have a unique identifier and a unique type. The identifier corresponds to the symbols C1, C2, ..., CN as used in the functional model. A contact is defined between exactly two parts; they are named as *front* and *back* parts of the contact (these terminologies are borrowed from the functional approach). There are four types of contacts used in our approach. They are: *planars, cylindrical, threaded-cylindrical* and *polyhedral*. For encapsulation purposes, a contact also maintains information about parts joined by the contact, attachments acting on the contact and relationships in which the contact is involved. Another useful structural parameter is the area of the contact which may later be used in mechanical stability and feasibility analysis.

Behaviour:

We do not include any operation at present in this object, because we do not perform engineering calculation. However, when required, operations to this class can be added.

Definition:

```
CLASS CONTACT {
data members:
  identifier: String;
  type: {planar, cylindrical, threaded-cylindrical, polyhedral}
  forward-or-along-part: PART
  backward-or-against-part: PART
  contactArea: REAL
  relatedAttachments: set of ATTACHMENT
  relatedRelationships: set of RELATIONSHIP
};
```

3. CLASS ATTACHMENT

Structure:

Like parts and contacts, each attachment belonging to the class ATTACHMENT must have a unique identifier and a unique type. We consider the following four types of attachments: *SCREW, GLUE, CLIP* and *PRESSUREFIT*. This list can be

extended without affecting the algorithms or the model. A attachment has a weight limit that it can withstand.

Each attachment acts on only one contact. Correspondingly, a relationship is established between the parts involved in the contact, the contact itself and the attachment.

Behaviour:

We do not include any operation at present in this object, because we do not perform engineering calculation. However, when required, operations to this class can be added.

Definition:

```
CLASS ATTACHMENT {
data members:
  identifier: String
  type: (GLUE, SCREW, CLIP, PRESSUREFIT)
  contactActingOn: CONTACT
  relatedRelationships: set of RELATIONSHIP
  maxWeight: REAL
};
```

4. CLASS RELATIONSHIP

Structure:

A relationship in this approach is always binary and is applied between two components chosen from objects of classes PART, CONTACT and ATTACHMENT. Every relationship is uniquely identified by a distinct identifier such as R1, R2, ..., RN as used in the functional approach. We consider the following types of relationships in our approach: *part-contact relationship, agent-attachment relationship* and *target-attachment relationship* (these names are derived from work done using the functional approach). In the class RELATIONSHIP, we include three attributes referring to parts, contacts and attachments on which a relationship is defined. Since any instance of this class uses exactly two of these attributes, the third attribute is set to NIL. For example, in a *part-contact relationship*, the attachment attribute is set to NIL.

Behaviour:

We do not include any operation at present in this object, because we do not perform engineering calculation. However, when required, operations to this class can be added.

Definition:

```
public class RELATIONSHIP {

data members:

  identifier: String

  type: {part-contact, target-attachment, agent-attachment, blocking-part-attach-
        ment}
```

relatedPart: PART

relatedContact: CONTACT

relatedAttachment: ATTACHMENT

AgentOrTargetName: String

};

We now consider the *a-functions* and their categories described for the functional approach in Chapter 2. Below, in Table 5, we illustrate that these categories are already captured by the class definitions.

TABLE 5.

a-function in category	captured by the class
Category 1	PART
Category 2	CONTACT
Category 3	CONTACT
Category 4	CONTACT
Category 5	CONTACT
Category 6	CONTACT
Category 7	ATTACHMENT
Category 8	RELATIONSHIP
Category 9	PART and RELATIONSHIP
Category 10	CONTACT and RELATION-SHIP
Category 11	1) ATTACHMENT and RELATIONSHIP 2) CONTACT and RELA-TIONSHIP
Category 12	1) ATTACHMENT and RELATIONSHIP 2) CONTACT and RELA-TIONSHIP
Category 13	RELATIONSHIP
Category 14	RELATIONSHIP

So far, we have captured all the input information from the relational model. In addition, our object-oriented model also requires other classes in order to generate the AND/OR graph. These are described below:

5. CLASS ASSEMBLY

Structure:

An assembly consists of a set of parts, a set of contacts, a set of attachments and relationships among these. This is a major class in our model incorporating the important functionalities required to generate AND/OR graph.

Behaviour:

Given an assembly, one can generate all its feasible decompositions and can generate its corresponding AND/OR graph.

As stated earlier, consistency of information among the parts, contacts, attachments and relationships must be established. This is described by the function *data-consistency( ).*

All the parts in an assembly should be connected through the contacts. This check is described by the function *isConnected( ).*

Definition:

public class ASSEMBLY {

data members:

```
  parts: set of PART
  contacts: set of CONTACT
  attachments: set of ATTACHMENT
  relationships: set of RELATIONSHIP
  andorgraph: ANDORGRAPH
member functions:
  public set-of-DECOMPOSITION Gen-feasible-decompositions()
  public void GENERATE_AND_OR_GRAPH();
  public boolean data-consistency();
  public boolean isConnected();
}
```

6. CLASS DECOMPOSITION

Structure:

Each decomposition consists of two subassemblies

Behaviour:

An important behavioural component of a decomposition is to check its feasibility.

Definition

```
public class DECOMPOSITION {

data members:

  sub-1: ASSEEMBLY

  sub-2: ASSEMBLY

member functions:

  public boolean feasible-test();

}
```

7. CLASS NODE

Structure:

This class represents a node in the AND/OR graph. It corresponds to one sub-assembly during the assembly process. The possible OR compositions of this sub-assembly is also included in the node.

Behaviour:

None

Definition:

```
public class NODE {

data members:

  subassem: ASSEMBLY
```

OR-arcs: set of DECOMPOSITION

};

8. CLASS ANDORGRAPH

Structure:

An AND/OR graph consists of set of NODEs.

Behaviour:

The two relevant operations for an AND/OR graph are: "*add*" to add a node during the generation of the graph and "*print*" to print the graph in a nice format.

Definition:

```
CLASS ANDORGRAPH {

data members:

  nodes: set of NODE;

member functions:

  void add(node);

  void print();

};
```

We do not define *SUBASSEMBLY* as a separated class since it performs the same functionality and attributes as that of ASSEMBLY; in other words, a SUBASSEMBLY is an

ASSEMBLY. Hence, these two terms are used interchangeably. The following shows the object model of these classes (Figure 5).

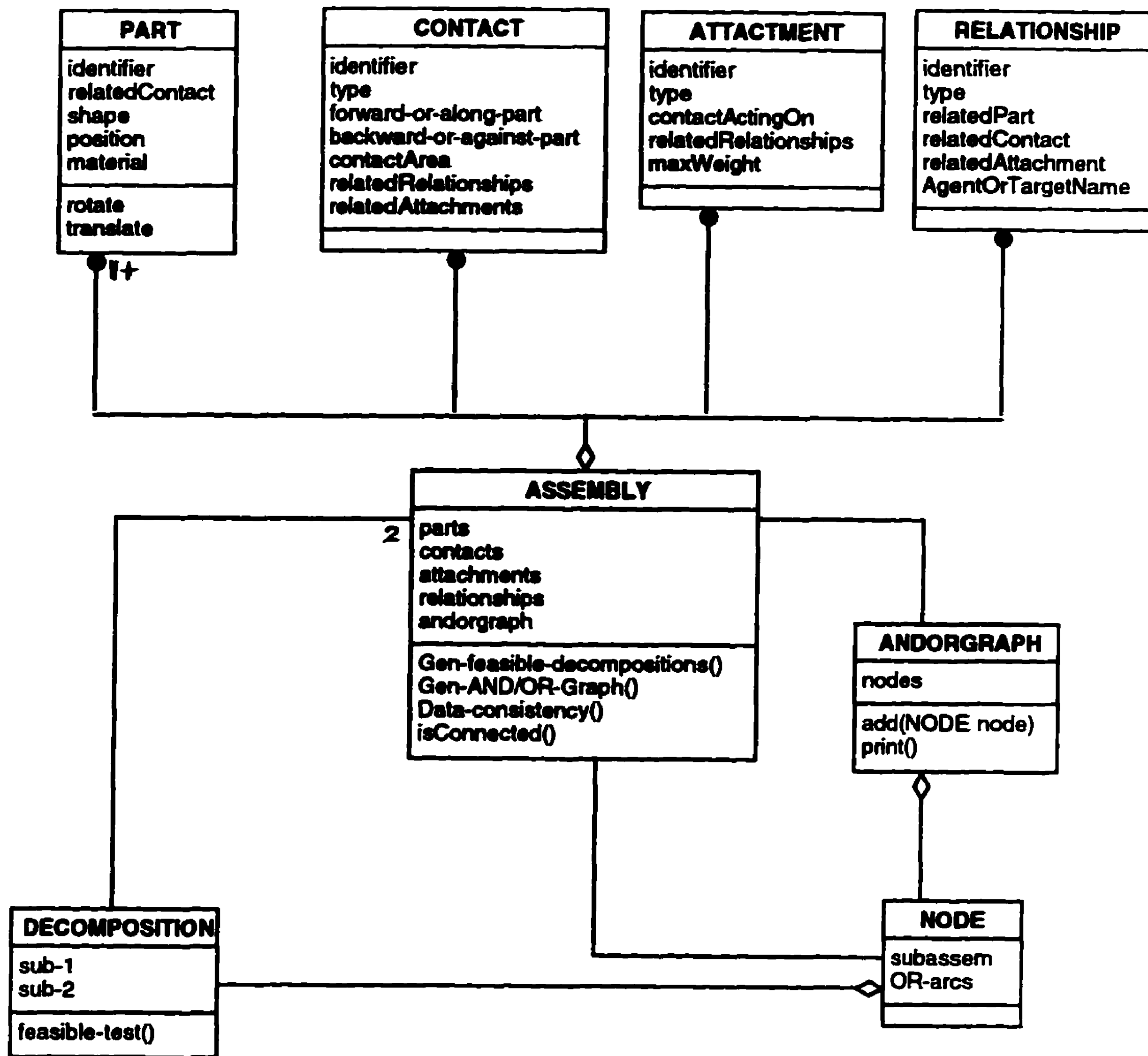


Figure 5 Object Model

3.3 Interactions among the objects

Having defined the classes, we now show how the objects from these classes interact with each other to generate the AND/OR graph.

The basic idea of generating the AND/OR graph for an assembly is to take the information of the parts in the assembly, find all possible decompositions each resulting in two sets of connected parts, and then check the feasibility of these decompositions. For each decomposition which passes the feasible-test, we must create a node in the AND/OR graph. By appropriately connecting each decomposition node to its descendants, we will complete the AND/OR graph.

3.3.1 Generate the Assembly

One of the prime requirements of the AND/OR graph generation method is to ensure consistency of information among the entities. For example, when two parts are joined together to make a subassembly, they must have same or compatible shapes. In our model, we do not provide additional methods for such mechanical compatibility; rather, we assume that such algorithm could be easily introduced at the implementation level. The other consistency issue in the object-oriented approach is due to the redundant information stored in each class. For example, a part includes the set of contacts in which the part is involved while a contact includes the two parts making up the contact. It is therefore required to ensure consistency among these duplicated information.

We describe these checks in the method *data-consistency()* which consist of four phases. The flowcharts shown in Figure 6 - 9 illustrate these four phases. In all the flowcharts in this thesis, a solid arrow represents data flow and dash-arrow indicates control flow.

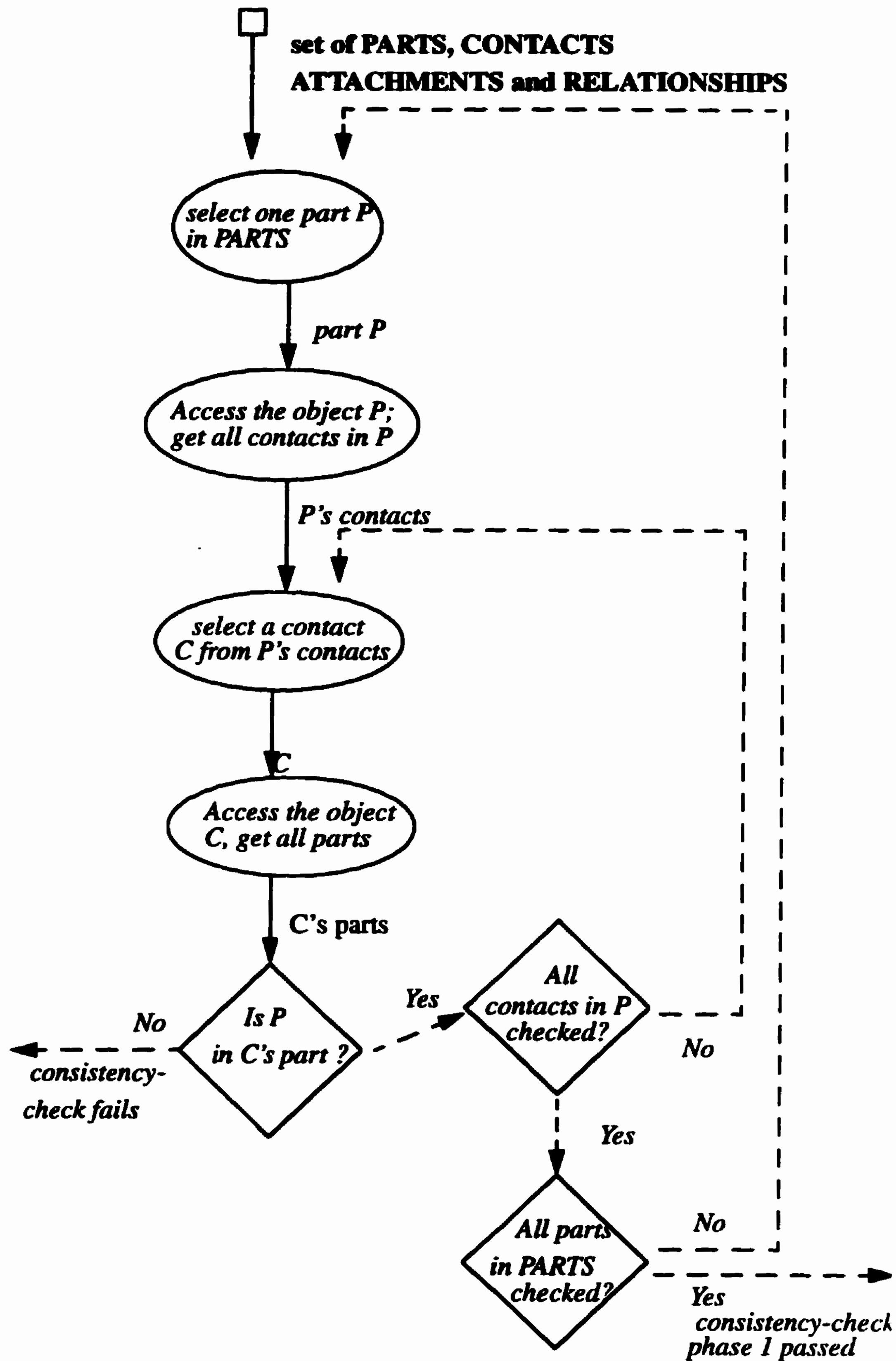


Figure 6 data-consistency- Phase 1

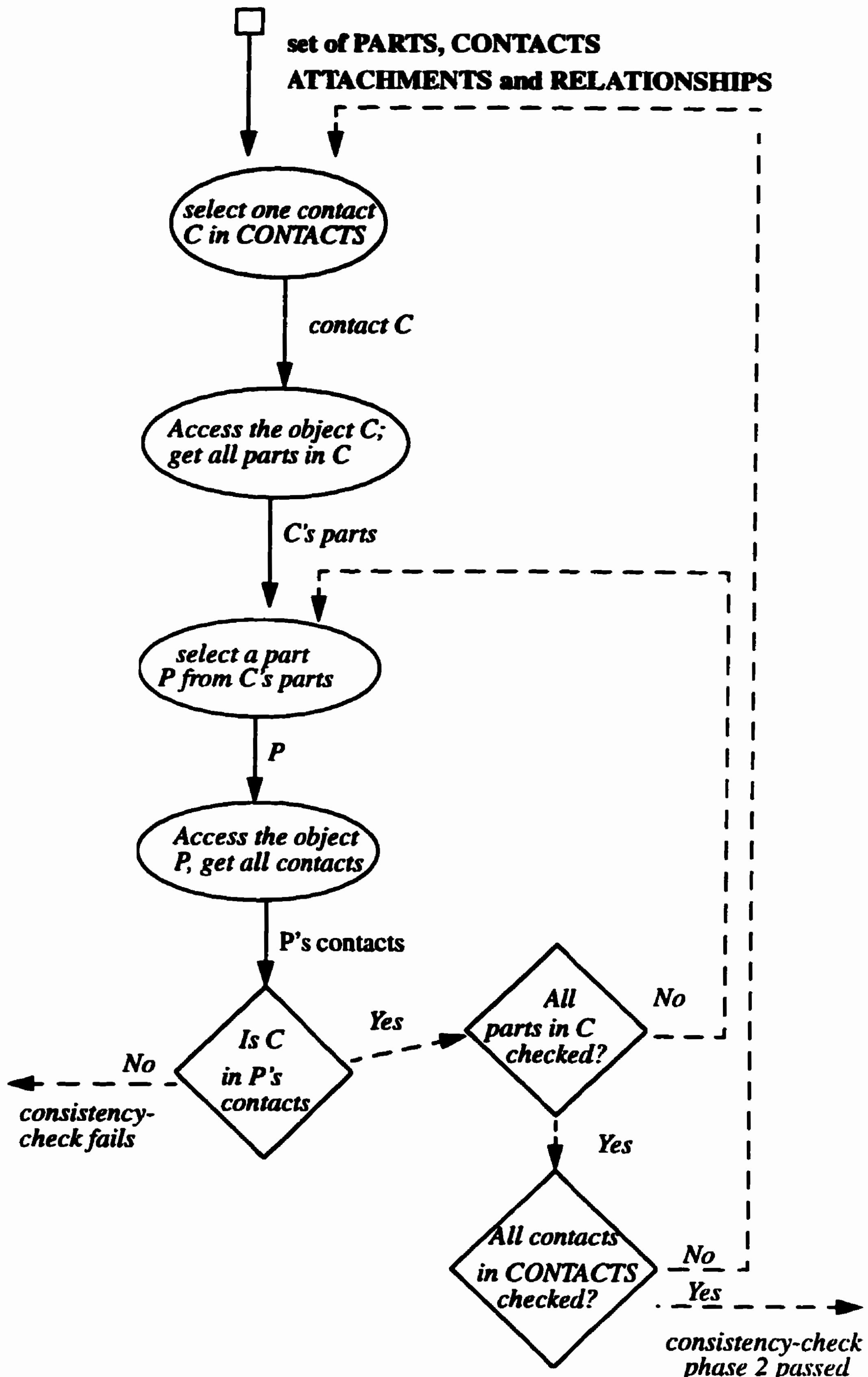


Figure 7 data-consistency phase 2

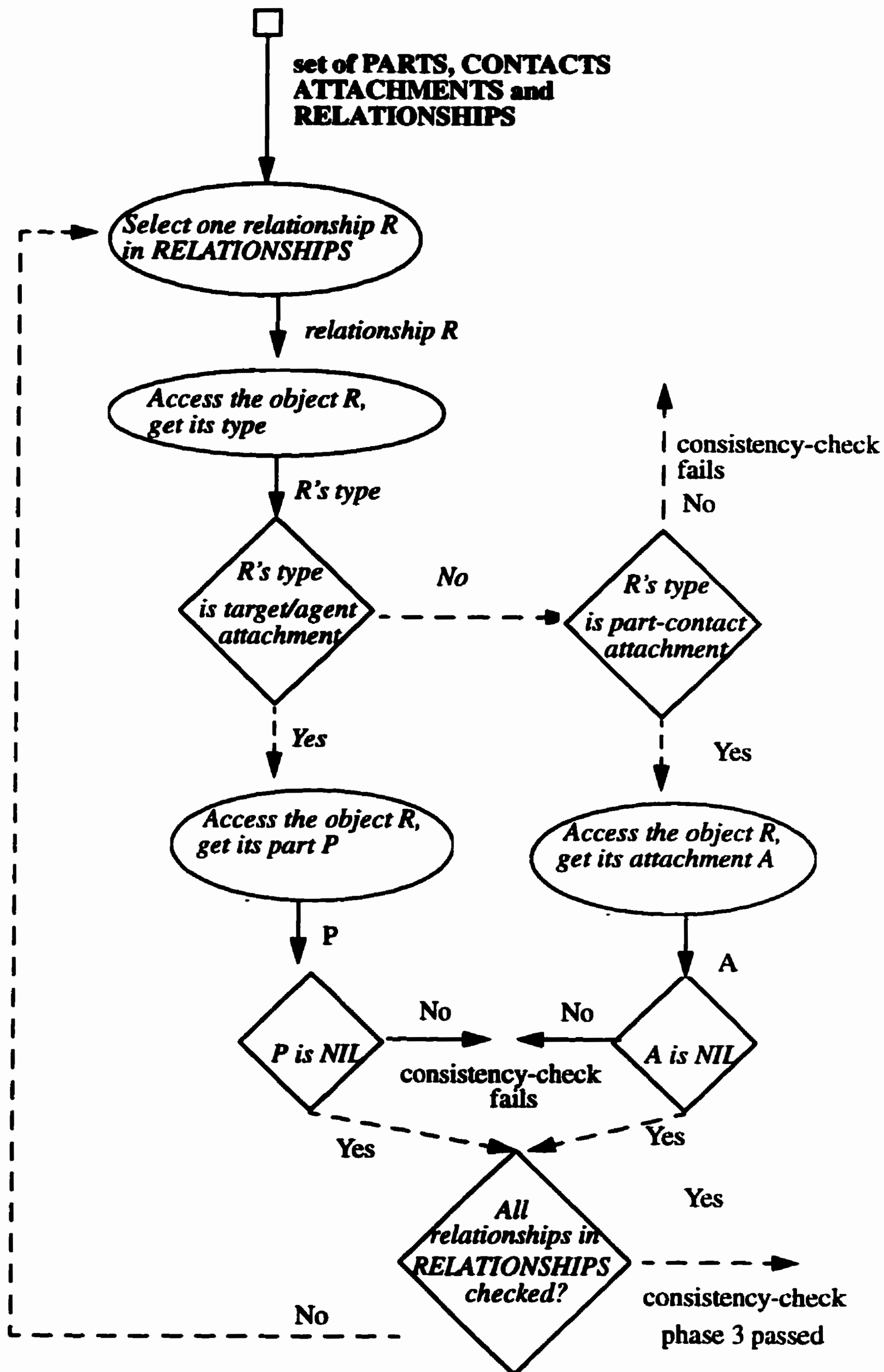


Figure 8 data-consistency- phase 3

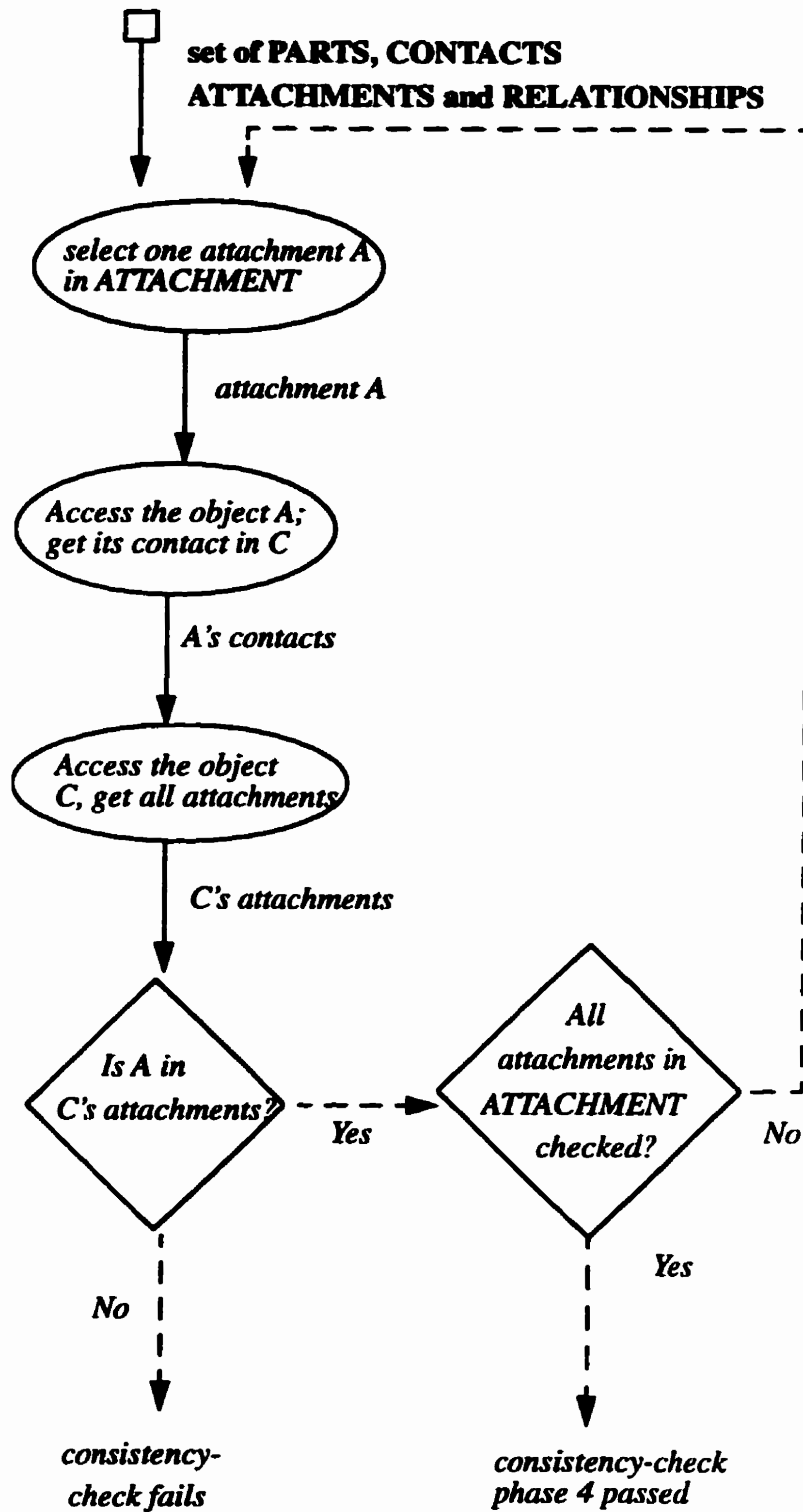


Figure 9 data-consistency - phase 4

3.3.2 Generate Feasible Decompositions

The next step is to generate the feasible-decompositions for the assembly.

The functional approach includes two data structures during the generation of feasible decompositions: ***cut-sets*** and ***connection-graph***. A connection graph is a simplified version of the relational graph input to the assembly sequence generation process and hence can be safely ignored. As observed from the paper [9], the connection graph is used only to generate the cut-sets. In our approach, we ignore the cut-sets itself. The justification is that a cut-set is a temporary data structure which is used to identify the two subassemblies during a decomposition process. We propose to model "decomposition" as a separate class which therefore includes the concept of cut-set.

The generation of feasible decomposition proceeds as follows: Starting with a completely assembled product, we generate two assemblies by breaking a set of contacts. In this context, we still follow the notion of a cut-set, but without mentioning or storing it. Each of these subassemblies is subject to (i) a connectivity test which ensures whether the parts are geometrically connected, and (ii) to a feasibility test which ensures that the parts can be mechanically joined together to form a stable subassembly.

Figure 10 shows the flowchart of the algorithm Gen-feasible-decompositions().

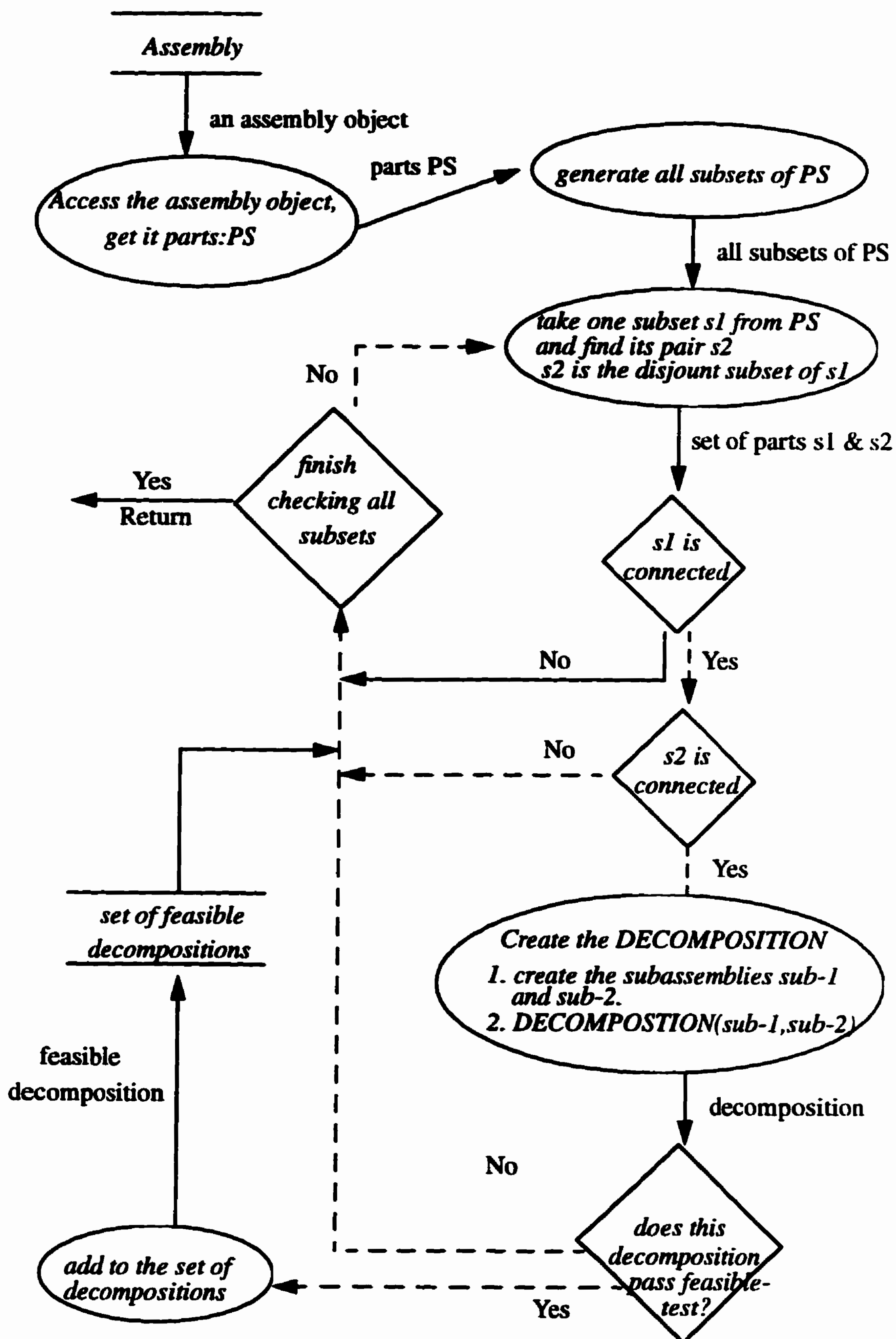


Figure 10 Gen-feasible-decompositions()

In Gen-feasible-decompositions method showed above, the ASSEMBLY class interacts with the following classes:

- ASSEMBLY class: the method *isConnected()*

- DECOMPOSITION class: the method *DECOMPOSITION*() to create a DECOMPOSITON object

- DECOMPOSTION class: the method *feasible-test*()

Each of these methods are described in detail below:

isConnected

This algorithm determines whether or not a given set of parts are connected through the contacts specified in the assembly. Let S_{in} be the set of parts under consideration. The algorithm starts by selecting a part P in S_{in}, access all the contacts in P and includes all the parts that are connected to P through the contacts in another set S_{temp}. This process is repeated for every part in S_{temp} when all the parts in S_{temp} are analyzed, the algorithm checks whether S_{in} and S_{temp} are identical. If so, it confirms that the set of parts S_{in} are all connected. Figure 11 illustrates this algorithm through a flowchart.

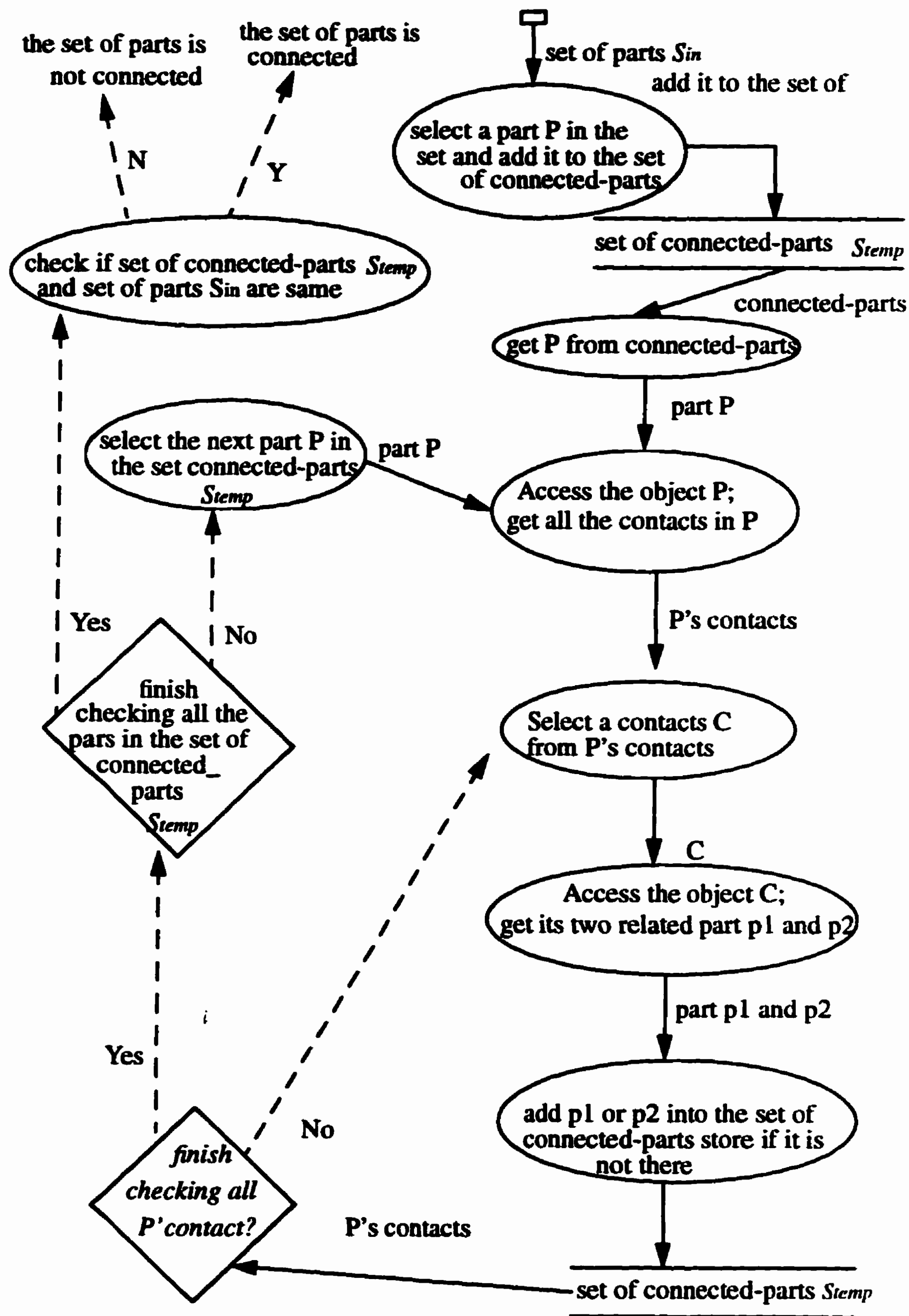


Figure 11 isConnected()

DECOMPOSITION():

This algorithm represents a constructor function. Given two sets of parts, this algorithm constructs an instance of the class DECOMPOSITION (see the class definition DECOMPOSITION). An instance of DECOMPOSITION requires two subassemblies. The information for creating each subassembly can be derived from the sets of parts. The rest of the algorithm is trivial.

feasible-test()

This algorithm determines whether or not a decomposition is a valid decomposition. In the context of AND/OR graph generation process, this refers to checking whether the reverse of a disassembly process is feasible. The functional approach uses two criteria to determine the feasibility: stability of the two subassemblies and mechanical feasibility of the two subassemblies. The former refers to judging whether each subassembly is stable by itself and the latter ensures whether there exists a mechanical process by which the two subassemblies can be joined together. In our approach, we present all the information regarding the two subassemblies to the user and let the user decide the feasibility. Hence, we do not describe this algorithm any further in this thesis.

3.3.3 Generate AND/OR Graph

Once the feasible decompositions generated, we can now develop the AND/OR graph. As given in the class ANDORGRAPH, an AND/OR graph consists of a set of nodes; each node indicates the subassembly and its all feasible decompositions. Hence, it is a straightforward task to generate the graph using the set of decompositions. In the implementation, the generation of AND/OR graph is done recursively whenever a decomposition is identified, we discuss this in detail in the next chapter.

3.3.4 Example:

Now we illustrate the generation of an AND/OR graph using the example shown in Figure 1.

Step 1. Create the assembly object from the set of PART, set of CONTACT, set of ATTACHMENT and the set of RELATIONSHIP.

• input:

parts = {P1, P2, P3, P4, P5}

contacts = {C1, C2, C3, C4, C5, C6, C7, C8}

attachments = {A1, A2, A3, A4, A5}

relationships = {R1, R2, R3,..., R26}

• Check consistency of the input data.

This is done by calling the data-consistency() algorithm.

• Create an assembly object using the constructor function in ASSEMBLY

ASSEMBLY: ASSEMBLY(parts, contacts, attachments, relationships)

STEP 2: Create the feasible decomposition list

• Create all subset of parts and pair them. The list of all pairs is given below:

{p1}{p2 p3 p4 p5}	{p2} {p1 p3 p4 p5}	{p1 p2} {p3 p4 p5}
{p3} {p1 p2 p4 p5}	{p1 p3} {p2 p4 p5}	{p2 p3} {p1 p4 p5}
{p1 p2 p3} {p4 p5}	{p4} {p1 p2 p3 p5}	{p1 p4} {p2 p3 p5}

{p2 p4} {p1 p3 p5} {p1 p2 p4} {p3 p5} {p3 p4} {p1 p2 p5}

{p1 p3 p4} {p2 p5} {p2 p3 p4} {p1 p5} {p1 p2 p3 p4} {p5}

- Check whether each pair represents a set of connected parts. This results in the following list:

{p1}{p2 p3 p4 p5} {p2} {p1 p3 p4 p5} {p1 p2} {p3 p4 p5}

{p3} {p1 p2 p4 p5} {p4} {p1 p2 p3 p5} {p1 p4} {p2 p3 p5}

{p1 p2 p4} {p3 p5} {p3 p4} {p1 p2 p5} {p1 p3 p4} {p2 p5}

{p1 p2 p3 p4} {p5}

- For each pair, create an instance of DECOMPOSITION.
- Perform feasibility test on each decomposition and delete those which fail the test. The resulting list after the feasibility test is:

{p3 p4 p5} {p1 p2} {p3} {p1 p2 p4 p5} {p3 p5} {p1 p2 p4}

{p1 p2 p5} {p3 p4}

Step 3: Generate AND/OR graph

- The result is shown in Figure 4.

3.4 Formal description of the Object-Oriented approach

We now discuss the formal, abstract object-oriented model of an assembly. The notations used in this section are listed in Appendix A.

A part is characterized by a unique part name and a set of contacts in which the part is involved. Initially, this set of contacts will be empty and during the assembly process, it will be updated. Formally,

$$\text{Part} == PartName \times \prod (Contact)$$

$$\text{Initial: } \prod = \emptyset\,(Contact)$$

A contact is characterized by its contact name and the two parts involved in that contact.

$$Contact == ContactName \times PartName \times PartName$$

An assembly is described by a set of parts making up the assembly. This model of assembly coincides with the functional model so that it is easy to compare them. The set of contacts in the assembly can be extracted from the parts records. Initially, an assembly is empty.

$$Assembly == \prod (Part)$$

$$\text{initial: } \prod (Part) = \emptyset$$

A subassembly is an assembly by itself. Therefore, it simply inherits the definition of an assembly.

Subassembly inherits Assembly

A graph of connection for an assembly consists of a set of parts and a set of contacts.

$$Graph == \prod(Part) \times \prod(Contact)$$

Having defined the model, we now establish several invariant conditions on the model. These invariant conditions assert the validity of each entity whenever it is created or used. For example, the invariant of an assembly asserts that the assembly should be stable and valid (using the *sa* and of predicates in the functional model. Following give the invariant conditions for the various entities defined above:

- For every contact *c* within a part's record p, the name of *p* must be stored in the record of *c*. This ensures the consistency of information between the contacts and parts. In a similar way, for every part name *pn* in a contact *c*, the contact *c* must be stored in the corresponding part's record (whose name is *pn*).

$$\forall p : Part \bullet \forall c : Contact \bullet \; c \in \prod(Contact)\,(p)$$

$$\Rightarrow PartName\,(p) \; = \; PartName\,(c)$$

$$\forall c : contact \bullet \forall p : Part \bullet PartName\,(p) \; = \; Partname\,(c)$$

$$\Rightarrow c \in \prod(Contact)\,(p)$$

We also need an additional constraint asserting that the two parts involved in a contact must be different. In other words, a contact cannot be established within the same part. Thus:

$$\forall c : Contact \bullet \exists (p_1, p_2) : Part \bullet PartName\,(p_1) = PartName\,(c) \wedge$$

$$PartName(p_2) = PartName(c) \Rightarrow p_1 \neq p_2$$

- Every assembly must be stable and valid.

$$\forall a : Assembly \bullet st(a) \bullet sa(a)$$

The predicates "*st*" and "*sa*" have the same interpretations as in the functional model (i.e. stability and validity)

A single part constitutes an assembly (which is, by the previous invariant, valid and stable). The contacts in that part must be empty.

$$\forall p : Part \bullet \prod (Contact)\ (p) = \varnothing \Rightarrow \exists a : Assembly \bullet \prod (Part)\ (a) = \{p\}$$

- A subassembly should respect all invariants of an assembly. This is vacuously true by the semantics.

- The set of parts and the set of contacts in a graph of connection should be consistent with each other; in other words, the set of contacts in the record of every part in a graph of connection should contain only those contacts which are defined within the graph of connection and nothing else. Similar constraints apply to the set of contact.

$$\forall g : Graph \bullet$$

$$\forall p : Part \bullet p \in \prod (Part)\ (g) \Rightarrow \prod (Contact)\ (p) \subseteq \prod (Contact)\ (g)$$

$$\forall c : Contact \bullet c \in \prod (Contact)\ (c) \Rightarrow \exists p : Part \bullet$$

$$Partname(p) = PartName(c) \wedge p \in \prod(Part)(g)$$

An assembly process is now defined as an operation in (in object-oriented terminology, a behaviour of) *Assembly*. It typically merges two subassemblies into an assembly, making all the contacts between the parts involved. Formally it is defined as:

Assembly Process: $Assembly \times Assembly \rightarrow Assembly$

i.e., it is defined as a function taking two assemblies (or subassembly) and returning a composite. If *a1* and *a2* are the two operands for the assembly process, then it returns a third assembly *a* satisfying the following conditions:

- The assembly process must be mechanically feasible and geometrically feasible
- The set of parts in *a* must be the union of parts from *a1* and *a1*
- The set of contacts made during the assembly are derived from the graph of connections and the parts' records are updated accordingly.

$$AssemblyProcess(a_1, a_2) = a \Leftrightarrow mf(AssemblyProcess(a_1, a_2)) \wedge$$

$$gf(AssemblyProcess(a_1, a_2)) \wedge$$

$$\prod(Part)(a) = \prod(Part)(a_1) \cup (Part)(a_2) \wedge$$

Let $g = Graph(a)$ in

$\forall c: Contact \bullet c \in \prod(Contact)(g) \Rightarrow \exists p_1, p_2$

$Part \bullet P_1 \in \prod(Part)(a1) \wedge p_2 \in \prod(Part)(a_2) \wedge \Pi(Contact)(p_1) \leftarrow$

$\prod(Contact)(p_1) \cup \{c\} \wedge \Pi Contact)(p_2) \leftarrow$

Chapter 4

Comparison

In this chapter, we critically compare the functional and the object-oriented approaches for generating the AND/OR graph.

4.1 The equivalence of the two approaches

Basically, there are two ways to prove the equivalence of the two approaches: (i) For a given assembly, it can be shown that both the models generate the same set of assembly sequences. This approach is called validation in terms of software engineering terminology and it requires a lot of test cases (different assemblies) to convince that the two models are equivalent. (ii) The other approach is to prove that the two models are *homomorphic*; i.e., any operation performed on the two models will result in the same state, provided that the state of the two models are identical before performing the operation. The second approach is used here.

We claim that the two models are *homomorphic* based on the following facts:

- First, it is required to show that the object-oriented model has all the structural information required for the assembly task. In the functional model, each assembly task checks whether the subassemblies are stable and valid, and also checks whether the task is mechanically feasible and geometrically feasible. These checks use the parts records and contacts records which are available globally in the functional model.

In the object-oriented approach, the *a-functions* are localized within the class PART, CONTACT, ATTACHMENT and RELATIONSHIP and are used whenever they are required. Since parts contain all the information about those contacts in which the parts are involved. Access to parts information automatically provide access to those contacts as well; contacts contain all the information about the parts, attachments and relationships in which the contacts are involved, access to contacts information automatically provide access to those parts, attachments and relationships as well. Thus, the global information for an assembly task is distributed across the classes PART, CONTACT and ATTACHMENT and RELATIONSHIP which are used when needed. This indicates that for every assembly state in the functional model, one could easily derive the corresponding state in the object-oriented model. The justification for the existence of such a formal derivation can be found in [18].

- The assembly process is a local operation to the object Assembly (which is also applicable to subassembly through inheritance). During the assembly process, the conditions such as mechanical feasibility and geometric feasibility are checked as in the functional mode. Besides the functional model also ensures that the resulting assembly (or subassembly) is stable and valid. These two constraints are coded as local to the assembly class which are to be satisfied by every instance of the assembly class. Thus, all the four constraints mentioned in the functional model are taken care of by the assembly operation in the object-oriented model.

Thus, every invocation of the assembly process will result in the same state as defined in the functional model, and thus establishing the equivalence between the two models.

4.2 Advantages of the our approach over Homen de Mello and Sanderson's approach

We claim that the object-oriented model has several advantages over the functional model. These advantages are summarized below:

- The object-oriented model naturally fits into assembly sequence problem. As one of the characteristics of object-oriented design, our object-oriented model closely resembles the real-world application. For example, the DECOMPOSTION class includes two sub-assemblies which are being created during a decomposition process. Another characteristic of the object-oriented model is that the model provides an abstraction of the real-world as well as a mechanism to implement or to realize the abstraction. Thus, the object-oriented approach fits into the assembly sequence problem because the assembly sequence problem deal with physical objects and their interactions,

- The object-oriented model provides the flexibility for easy extension and maintenance of the software derived from the model. Due to the separation of abstraction and implementation, one can change the implementation without affecting the abstraction. For example, the current implementation of the *feasible-test* method is interactive, letting the user to decide the feasibility, we can automate this process by modifying the code for the class DECOMPOSITION alone. This modification does not require recompilation of other classes. Due to high information hiding characteristics of the class, one can also enhance a class representation without affecting the rest of the system. As an example, one can add more information to the class PART which might include functionality of the part in the overall product.

- Our approach eliminates the redundant algorithms and data structures in the functional model. The two major data structures eliminated are *CUT-SET* and *CONNECTION-GRAPH*. This make the whole system simpler and easier to understand.

• In our approach, we have changed the non-recursive algorithms. In the functional model to recursive algorithms. This make the algorithms easier to understand and easier to implement.

4.3 Limitations of the approaches

Both approaches have limitations on the computer resources when there are a lot of parts in the assembly. Larger assemblies are quite common in the automobile industries. This is an inherent problem of the AND/OR graph method itself and not on the approaches. The amount of computation involved in generating all mechanical assembly sequences was assessed by determining the number of decompositions that must be analyzed[9]. This will dramatically increase with the number of the parts in the assembly.

As suggested in [9], two strategies can be applied to address this problem:

1. Artificially reduce the number of parts by treating subassemblies as single parts.

2. The algorithm generate fewer, hopefully the best, sequences using some heuristics to guide the generation of assembly sequences. Such heuristics should be compatible with the evaluation function used to choose among the alternative assembly sequences[9].

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we have used the object-oriented approach to redesign Homen de Mello and Sanderson's assembly sequence generation algorithm, also known as AND/OR graph generation algorithm. We do not propose a new algorithm; rather, we redesigned the functional model using the object-oriented approach.

In our approach, we retain all information contained in first four elements of the quintuple <P, C, A, R, a-functions> through the class definitions PART, CONTACT, ATTACHMENT and RELATIOINSHIP. The information captured by the *a-functions* is distributed across the four classes.

In addition, we have introduced four other classes ASSEMBLY, ANDORGRAPH, DECOMPOSITON and NODE. Among these classes, ASSEMBLY is the major class incorporating the important functionalities required to generate the AND/OR graph.

We also give a formal model of the object-oriented design in this thesis and established the equivalence between the object-oriented model and the functional model. We claim that the object-oriented model is easy-to-understand and easy-to-enhance which are typical characteristics of the object-oriented paradigm.

5.2 Future work

With regard to possible extensions to this thesis, we propose that the object-oriented model can be strengthened in the following aspects:

1. The method *isConnected()* in class ASSEMBLY can be optimized. By clever techniques, we could modify this algorithm to reduce the number of accesses to objects of classes PART and CONTACT. The complexity of the algorithms in object-oriented approach generally depends on the number of accesses to individual objects.

2. The *feasible-test()* algorithm can be elaborated and partially automated by introducing algorithms to perform engineering calculations.

3. The run-time efficiency of the *Gen-AND/OR-graph()* method can be improved by eliminating unnecessary and redundant computations. One way is to check whether the subassembly is computed before. For example, when we generate the AND/OR graph for subassembly {p1 p2 p4}(node B in Figure 4), we found it was computed before (node A), therefore there is no need to do further computing; we can as well use the AND/OR graph from node A.

Appendix A

Formal Notations

$X == Y \times Z$

means *X* is a type, defined as a cartesian product of Y and A. If *x* is of type *X*, then $Y(x)$ and $Z(x)$ refer to the first and second component of *x* respectively.

$\in$ denotes set membership.

$\varnothing$ denotes emptyset.

$\subseteq$ denotes subset relationship.

$\prod(P)$ denotes the powerset of P.

$\{p\}$ denotes singleton set; a set with only one element.

$\wedge$ denotes logical AND (conjunction).

$\Rightarrow$ denotes logical implication.

$f: X \times Y \rightarrow Z$

means *f* is a function whose inputs are of types *X* and *Y* and whose output is of type *Z*.

$\forall x : X \bullet < \text{predicate} >$

denotes universal quantification; i.e., for all *x* of type *X*, < *predicate* > is true.

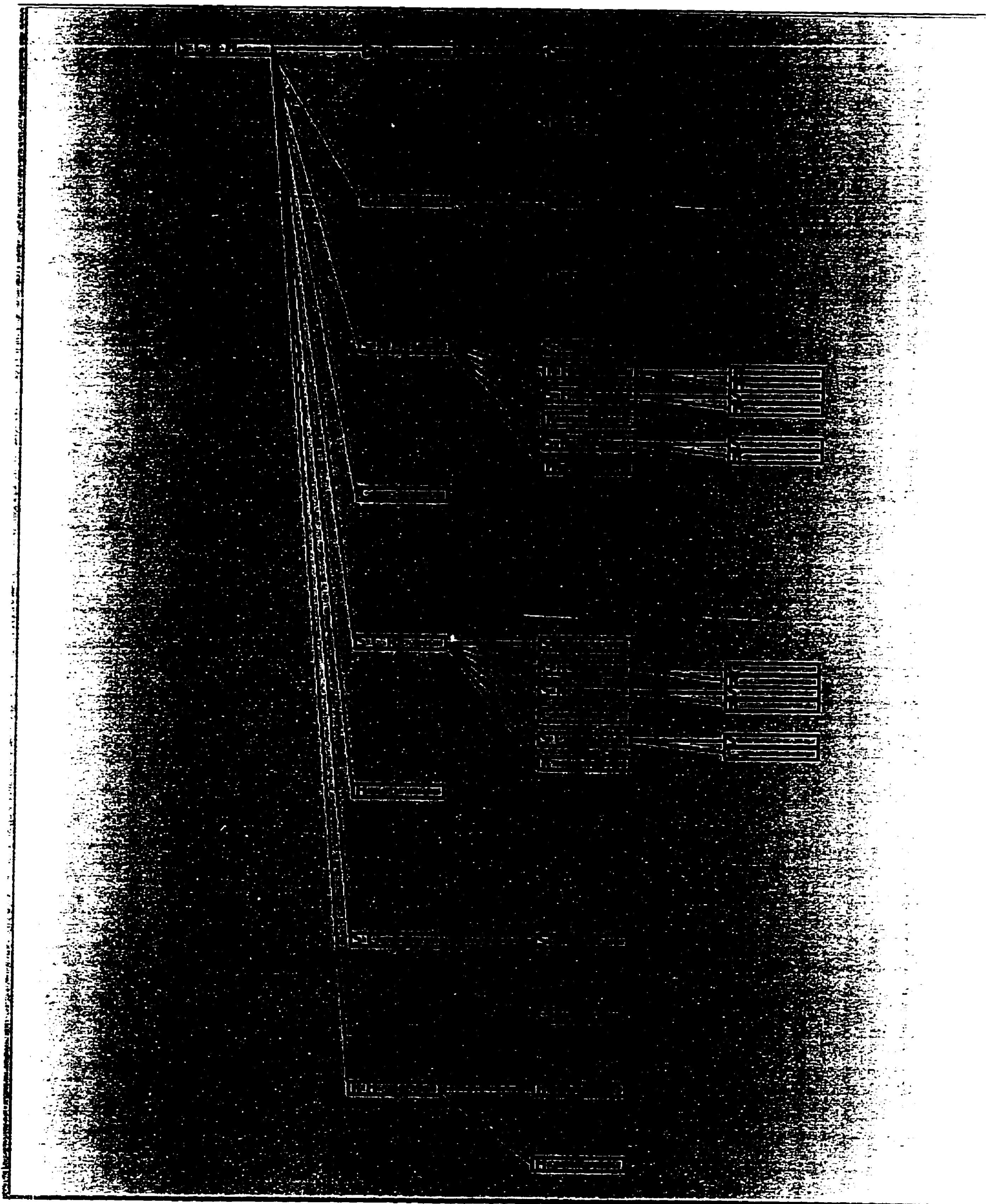
$\exists x : X \bullet < \textit{predicate} >$

denote existential quantification; i.e., there exists at least one *x* of type *X* for which < *predicate* > is true.

Appendix B

AND/OR graph for the example assembly

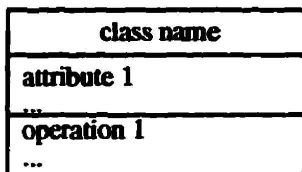
The following AND/OR graph is generated by the object-oriented approach for the example assembly in [9]



Appendix C

Object Model Notation

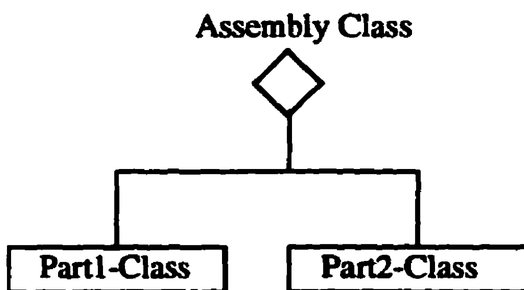
Class:



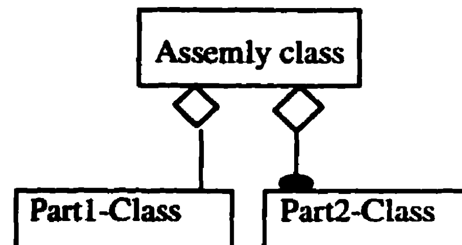
Association:

Multiplicity of associations:

Aggregation:



Aggregation (alternate form)



Bibliography

[1] D. F. Baldwin, T. E. Abell, C. M. Lui, T. L. De Fazio and D.E. Whitney, "An Integrated computer Aid for Generating and Evaluating Assembly Sequences for Mechanical Products", *IEEE Journal of Robotics and Automation*, 7(1):78-94, February 1991.

[2] G. Booth, *Object-Oriented Analysis and Design with Applications*, (Second Edition), Benjamin Cummings Publishers, 1994.

[3] T. Cao and C. Sanderson, "Task sequence planning in a robot workcell using and/or nets", *IEEE International Symposium on Intelligent Control*. pp. 239-244. 1991.

[4] G. Dini and M. Sanlochi, "Automated sequencing and subassembly detection in assembly planning", *CIRP Annals*, Vol. 41, No. 1, pp. 1-4. 1992.

[5] H. A. Elmauaghy and L. Laperriere, "Modeling and Sequence generation for robotized mechanical assembly", *Robotics and Autonomous Systems*, Vol. 9, No. 3, pp. 137-147, 1992.

[6] T. L. De Fazio and D. E. Whitney, "Simplified Generation of all Mechanical Assembly Sequences", *IEEE journal of Robotics and Automation*, Vol, RA-3, No. 6, pp. 640-658, Dec 1987.

[7] L. Hara and T. Nagata. "Robot assembly planning based on and/or graphs", *Flexible Automation*, pp. 1587-1590. 1992.

[8] L. S. Homem de Mello, "Multihierarchical representation of tetrahedral truss structures for assembly sequence planning", *IEEE Transactions on Robotics and Automation*, Vol. 3, No. pp. 2398-2403. 1992.

[9] L. S. Homem de Mello and A.C. Sanderson, "A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences", *IEEE International Conference on Robotics and Automation*, Vol. 7, No. 4, pp. 228-240, 1991.

[10] L. S. Homem de Mello and A C. Sanderson, "Representations of Mechanical Assembly Sequences". *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 2, April 1991.

[11] L. S. Homem de Mello and A. C. Sanderson, "Two Criteria for the Selection for Assembly Plans: Maximizing the Flexibility of Sequencing the Assembly Tasks and Minimizing the Assembly Time Through Parallel Execution of Assembly Tasks", *IEEE Transactions on Robotics and Automation*, Vol 7. No. 5, October 1991.

[12] Y. F. Huang and C. S. G. Lee. "A framework of knowledge-based assembly planning". *IEEE Transactions on Robotics and Automation*, Vol. 1. pp. 599-604. 1991.

[13] K. Huang, "Development of an Assembly Planner Using Decomposition Approach", *IEEE Transactions on Robotics and Automation*, pp. 63-68, May 1993.

[14] S. S. Krishnan and A. C. Sanderson, "Reasoning about geometric constraints for assembly sequence planning", *IEEE Transactions on Robotics and Automation*, Vol. 1. pp. 776-782. 1991.

[15] S. Lee, "Backward assembly planning with assembly cost analysis". *IEEE Transactions on Robotics and Automation*, Vol. 3. pp. 2382-2391. 1992.

[16] A. C. Lin and T. C. Chang, "Integrated approach to automated assembly planning for three-dimensional mechanical products", *International Journal of Production Research*, Vol. 31, No. 5, pp. 1201-1227. 1993.

[17] K. Periyasamy and C. Mathew, "Mapping a Functional Specification to an Object-Oriented Specification in Software Re-engineering", *ACM Annual Computer Science Conference*, Philadelphia, PA, Feb 18-21, 1996.

[18] K. Periyasamy and W. Hu, "Object-oriented Design of an Assembly Sequence Planner", *Proceedings of the International Conference on Intelligent Manufacturing*, Wuhan, China, pp. 91-98, June 1995.

[19] P. Pu. "An Assembly Sequence Generation Algorithm Using Case-based Search Techniques", *IEEE Transactions on Robotics and Automation*, pp. 2425-2430, 1992.

[20] J. Rumbaugh *et al*, *Object-oriented Modeling and Design*, prentice Hall 1991.

[21] M. Santochi and G. Dini, "Computer-aided planning of assembly operations. the selection of assembly sequences", *Robotics and Computer- integrated Manufacturing*, Vol. 9, No. 6, pp. 439-446. 1992.

[22] H. K. Toensholl, E. Menzel and H. S. Park, "Knowledge-based system for automated assembly planning". *CIRP Annals*. Vol. 41, No. 1, pp. 19-22, 1992.

[23] J. Tsao and J. Wolter, "Assembly planning with intermediate states", *IEEE Transactions on Robotics and Automation*, Vol. 1, pp. 71-76, 1993.

[24] J. J. Waarts, N. Boneschanscher and W. F. Bronsvoort, "A Semi-Automatic Assembly Sequence Planner", *International Conference on Robotics and Automation*, pp. 2431-2438, May 1992.