Modeling Automatic Assembly and Disassembly Operations for Virtual Manufacturing

Swee M. Mok, *Member, IEEE,* Chi-haur Wu, and D. T. Lee, *Fellow, IEEE*

***Abstract*—A system for evaluating products in their design phase has been developed for virtual manufacturing. It is integrated into a CAD/CAM environment to calculate cost for assembling and disassembling parts. In our earlier work, a generic assembly and disassembly model was developed to represent operations required for product manufacturing and de-manufacturing. To be useful, the model requires a method for translating high-level instructions from product designers into low-level assembly and disassembly instructions. This paper presents a set of rules for accomplishing this task. The developed rules are used for manipulating strings representing parts and handlers in binary assembly and disassembly operations. A telephone assembly and disassembly simulation is used to illustrate the developed system.**

***Index Terms*—Assembly, CAD/CAM, design, disassembly, modeling, virtual manufacturing.**

I. Introduction

PRODUCTS that are difficult to assemble will raise manufacturing cost. Moreover, products that are difficult to disassemble will also increase cost in the near future as manufacturers are asked to recycle their products by a more environmentally conscious society [1], [2]. To overcome this potential problem, new products must be designed for easy assembly and disassembly, especially when automation is desired. In our previous work [3], [4], we proposed a solution that integrated a virtual assembly and disassembly (VIRAD) model into a CAD/CAM system. It models the production system as a hierarchical workcell so that product designers can easily evaluate products for manufacturing and de-manufacturing. The system is unique in such a way that it uses a developed part-mating coding system, named structured assembly coding system (SACS) [5], for analysis. The SACS codes can be directly implemented in a force compliant robotic workcell for automated assembly operations. As shown in Fig. 1, products are first designed using a CAD system. In the design process, software tools are used to calculate total assembly and disassembly cost. This feedback enables a designer to identify problematic parts and to redesign them as needed so as to minimize total product cost. Finally, the actual manufacturing and de-manufacturing are executed using automated machines modeled by a virtual assembly and disassembly (VIRAD) system.

The system aspect of a VIRAD model is the focus of this paper. We will present our work in defining rules for generating and representing assembly and disassembly operations and components so that a CAD system can be easily linked to a virtual manufacturing system.

II. System Design

Accurate modeling and simulation of product assembly and disassembly operations are key features of the proposed VIRAD system. The foundation of VIRAD is a newly developed hierarchical model named the generic assembly and disassembly (GENAD) workcell. It models assembly and disassembly operations carried out inside a workcell using a developed structured assembly coding system (SACS) [5], [6], which is briefly described in Appendix A. SACS uses numerical codes to represent a set of well-defined assembly and disassembly procedures. By associating each SACS code with its operation cost, a product's total manufacturing cost can be estimated. Based on SACS codes, a GENAD workcell uses a binary *assembly and disassembly tree* to represent all assembly and disassembly operations and costs. The data generated from a CAD software is used to generate the product's binary part-merging tree. Key information required will be the principal axis and mating-interface's geometry of all parts. The former is needed for assembly task planning while the latter is used for assigning various SACS codes for part-mating operations. A GENAD workcell is designed to model a workcell environment including its parts, tools, fixtures, and manipulators. All objects in the workcell are grouped into *parts* and *handlers*. Parts are consumable items for making products. Handlers are tools in the workcell for manipulating the parts in order to assemble or disassemble a product. In order to assemble and disassemble parts with handlers in the workcell, various operations defined by our SACS codes are used.

In order to generate an assembly and disassembly tree in a GENAD workcell, a three-step process, as illustrated in Fig. 2, is defined. The first step is to obtain a high-level, assembly tree for the designed product from the CAD system. The product designer can generally provide this tree. The second step is to define various handlers required for executing part-to-part mating operations and then assign these handlers to each part. We will also assign SACS codes for all operations performed by these handlers. The initial assignment is based on a set of standard handlers such as manipulators, part grippers, and

Manuscript received January 6, 2000; revised March 19, 2001. This work was supported by the Motorola Center for Telecommunication at Northwestern University. D.T. Lee's research was supported in part by the National Science Foundation under Grant CCR-9731638, and by the National Science Council under Grant NSC-89-2213-E-001-012. This paper was recommended by Associate Editor J. Lee.

S. Mok is with the Motorola Advanced Technology Center, Motorola Labs, Motorola Inc., Schaumburg, IL 60196 USA.

C. Wu is with the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208 USA (e-mail: chwu@ece.nwu.edu).

D. T. Lee is with the Institute of Information Science, Academia Sinica, Taipei, Taiwan, R.O.C.

Publisher Item Identifier S 1083-4427(01)04573-8.

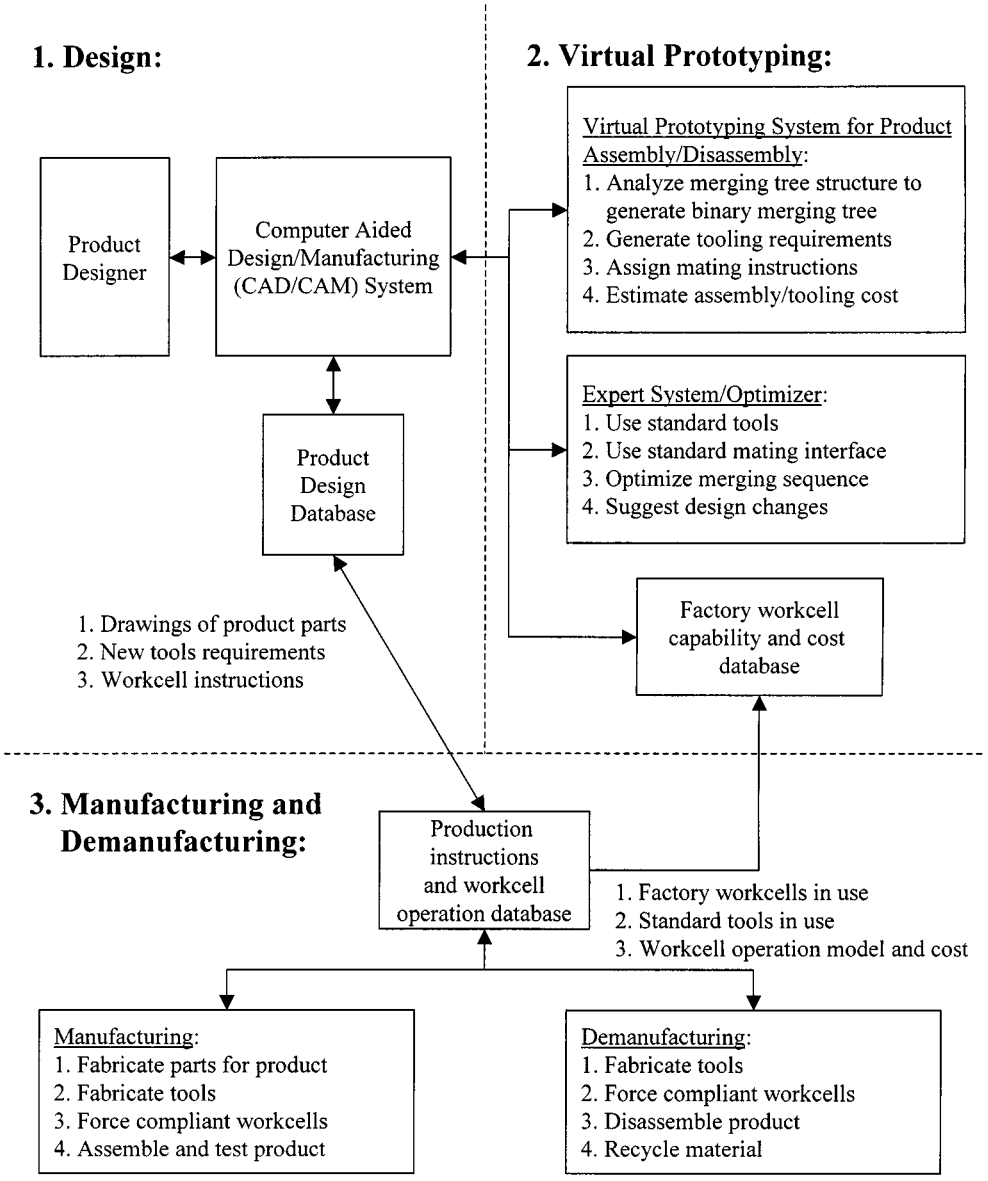

Fig. 1. Framework of proposed virtual assembly and disassembly for product design.

feeders, as shown in step 2 of Fig. 2. Re-using of handlers such as manipulators and part grippers in a workcell can then be applied to minimize cost. During the third step, assembly instructions for assembling the product will be generated from all part-to-part, handler-to-part, and handler-to-handler operations, as illustrated in step 3 of Fig. 2. Handler-to-handler operations are indirect operations associated with the assembly of a work piece, such as a robot attaching to a part gripper or a feeder being replenished with new parts. Together, these generated instructions form the product's assembly tree. In addition, each subpart's tree created from a disassembly operation is an independent binary tree representing operations that can be carried out in a parallel process. In Figs. 2 and 3, these relationships are represented using dotted lines.

From the assembly tree shown in Fig. 2, we can easily generate a disassembly tree shown in Fig. 3. We first rearrange the binary assembly tree from Fig. 2 in a top-to-bottom direction. For example, the assembly tree's root node becomes a leaf node in the disassembly tree. Then, the operators OP1 and OP2 in Fig. 2 are changed from '+' to '−' and '−' to "+', respectively. Thirdly, each SACS operator (in the assembly tree) associated with its parent node is now associated with its child node, as shown in part 1 of Fig. 3.

A GENAD workcell uses a small set of SACS-based operations to join primitive parts into larger and more complex parts during assembly, or vice-versa during disassembly. These are the lowest level instructions generated in this process. The rules for handling such operations will be described in the following section.

III. Assembly and Disassembly Operations

We shall adopt the following convention to describe assembly and disassembly operations and use strings to represent parts, resulting from these operations.

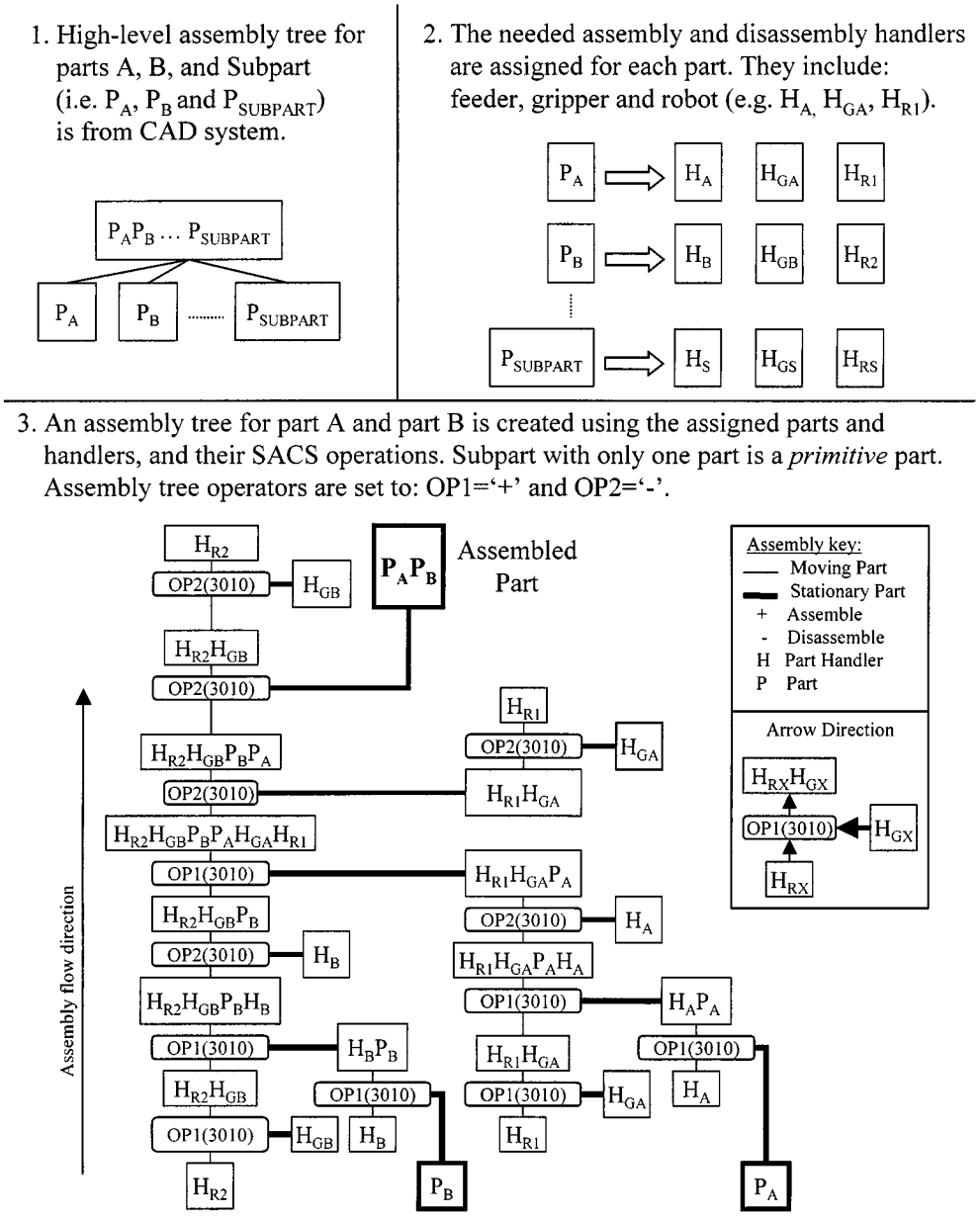

Fig. 2. Three steps to generate a GENAD tree from CAD data for assembling parts A and B.

Let Σ denote an alphabet set and *op* denote a binary operation, where $op \varepsilon \{+, -, o\}$ (symbol ε: "element of"). Let Σ^* denote a set of all composite sequences of elements in Σ. Associated with each operator $+, -$, there is a SACS code, and an associated cost. Moreover, each SACS code involves a *stationary* part (represented in **bold**) and a *moving* part.

A primitive part is denoted by $e \varepsilon \Sigma$ and a composite part resulting from a sequence of binary operations is represented by a string $S \varepsilon \Sigma^*$, where $\oslash \varepsilon \Sigma^*$ is a null object. We define string concatenation operation $\circ$ on strings S_1 and S_2 as $S_1 \circ S_2 = a_1 a_2 \ldots a_k b_1 b_2 \ldots b_m$ if $S_1 = a_1 a_2 \ldots a_k$, and $S_2 = b_1 b_2 \ldots b_m$, where $\{a_i, i = 1, 2, \ldots, k\} \varepsilon \Sigma$, and $\{b_j, j = 1, 2, \ldots, m\} \varepsilon \Sigma$. If $S = a_1 a_2 \ldots a_k$ then $(S)^r$ is defined as $S^r = a_k a_{k-1} \ldots a_1$. We further define $e \circ \oslash = \oslash \circ e = e$ for any primitive $e \varepsilon \Sigma$.

To assemble two parts $\mathbf{S_1}$ and $\mathbf{S_2}$ by *merging* S_1 with $\mathbf{S_2}$ we write $\mathbf{S_2}\{+\}S_1$ or $\mathbf{S_2 S_1}\{+\}$ and represent the resulting part as $(\mathbf{S_2} \circ S_1) = \mathbf{S_1 S_2^r}$. (We assume a stationary composite product represented in **bold**.) The former assembly notation is known as an infix expression, whereas the latter is known as a postfix expression. For example, "$\mathbf{ABDC} = \mathbf{CD}\{+(\text{SACScode})\}AB$" means that part $\mathbf{ABDC}$ is the resulting part of assembly operation $\mathbf{CD}\{+(\text{SACScode})\}AB$ in infix notation, or $(\mathbf{CD})(AB)\{+(\text{SACScode})\}$ in postfix notation.

To disassemble a part by *removing* a component from a composite part, there are two possible operations, $\{L-\}$ and $\{R-\}$ for left and right part removal operations, respectively. If S_1 is removed from the left of $\mathbf{S_1 S_2^r}$, we write $\mathbf{S_1 S_2^r}\{L-\}S_1$ or $(\mathbf{S_1 S_2^r})S_1\{L-\}$ and the resulting two parts are S_1 and $\mathbf{S_2}$. If S_2 is removed from the right of $\mathbf{S_1 S_2^r}$, we write $\mathbf{S_1 S_2^r}\{R-\}S_2$ or $(\mathbf{S_1 S_2^r})S_2\{R-\}$ and the resulting parts are also $\mathbf{S_1}$ and S_2. In our figures, $\{L-\}$ is treated as the default disassembly operation and it is represented using $\{-\}$.

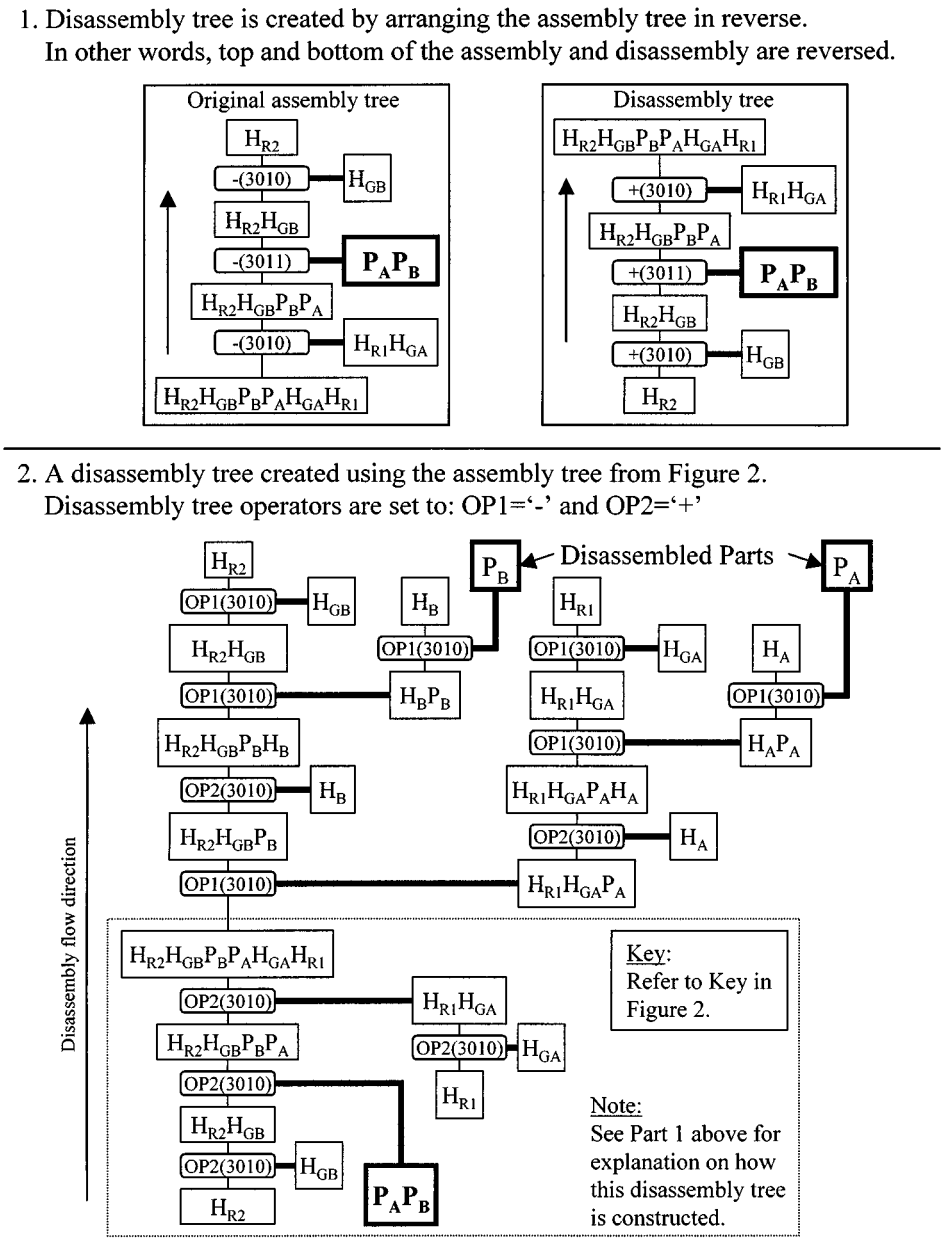

Fig. 3. GENAD binary tree generated for disassembling parts A and B.

For example, "$\mathbf{CD} = \mathbf{ABDC}\{-(\text{SACScode})\}AB$" means that part AB is removed from $\mathbf{ABDC}$ and the remaining part is represented by $\mathbf{CD}$, whereas "$\mathbf{AB} = \mathbf{ABDC}\{R - (\text{SACScode})\}CD$" will also result in two composite parts represented by $\mathbf{AB}$ and CD.

When multiple parts are involved in a sequence of assembly and disassembly operations, we may express it using an infix expression, following the conventional evaluation order of an arithmetic expression involving addition and subtraction operators and parentheses. For instance, $\mathbf{FA}\{+\}B\{+\}(\mathbf{CDE}\{-\}C)$ is evaluated as follows.

$$\begin{aligned}&\mathbf{FA}\{+\}B\{+\}(\mathbf{CDE}\{-\}C)\\&\quad=\mathbf{B}(\mathbf{FA})^r\{+\}(\mathbf{C}(\mathbf{ED})^r\{-\}C)\\&\quad=(\mathbf{BAF})\{+\}(ED)=(\mathbf{ED})(\mathbf{BAF})^r=\mathbf{EDFAB}\end{aligned}$$

We can also express the above sequence of operations using a postfix expression, which describes only the operations without explicitly giving the string representations of intermediate components. That is, $(FA)B\{+\}(CDE)C\{-\}\{+\}$. See [7] for details of how to evaluate a postfix expression.

In this example, if part C was removed first, the resulting part consists of two composite parts represented by ED and FA, and a primitive part represented by B. We then assemble part B with part $\mathbf{FA}$ to get $\mathbf{BAF}$. If now we choose $\mathbf{ED}$ as the stationary part, then by default, BAF becomes the moving object. Consequently, BAF will be placed to the left-hand side of $\mathbf{ED}$ to get a final product of $\mathbf{BAFDE}$—the reversed representation of $\mathbf{EDFAB}$. Although these two alphabetical strings look different, they still represent the same part in a physical world. As long as the mating interface between BAF and ED occurs between parts F and D, they are the same.

In the following we use P and H to denote a part and handler, respectively, and use subscripts to represent parts' and handler's names. For example, robot one and a part gripper for a part named A are represented by H_{R1} and H_{GA}. They are further illustrated in Fig. 2, whereby an object P_BP_A is assembled from two primitive parts P_A and P_B via an insertion operation. They are assembled using two robots, each equipped with a gripper. Robot number one (H_{R1}) starts by attaching a gripper for part A (H_{GA}) creating $H_{R1}H_{GA}$. The process is modeled as $\mathbf{H_{GA}}\{+(3010)\}H_{R1}$. SACS code 3010 is a position-controlled assembly operation with a cost c3010 to operate, defined in Fig.8. Robot H_{R1} is modeled by SACS code 3600, a six degrees of freedom (DOF) manipulator. H_{GA} is modeled by SACS code 3110, a one-axis gripper without force compliance, and is stationary during the assembly operation. All the other assembly steps in the same figure are self-explanatory following the aforementioned notation.

For the process of disassembling product P_AP_B into part P_A and part P_B, we examine the disassembly tree in part 2 of Fig. 3. The two robots with grippers used in the assembly process are again used to manipulate the parts. At the third level from the bottom of the disassembly tree in Fig. 3, they both grip onto P_AP_B. Following that, a disassembly operation between the interface of P_A and P_B takes place. Robot number one with a gripper proceeds to place P_A into a recycling bin H_A before disassembling itself from H_AP_A, representing the part and recycling bin. Similarly, robot two with a gripper proceeds to release part P_B into a recycling bin H_B. Finally, their grippers are returned to their holders at the top most level of the tree, completing the disassembly sequence.

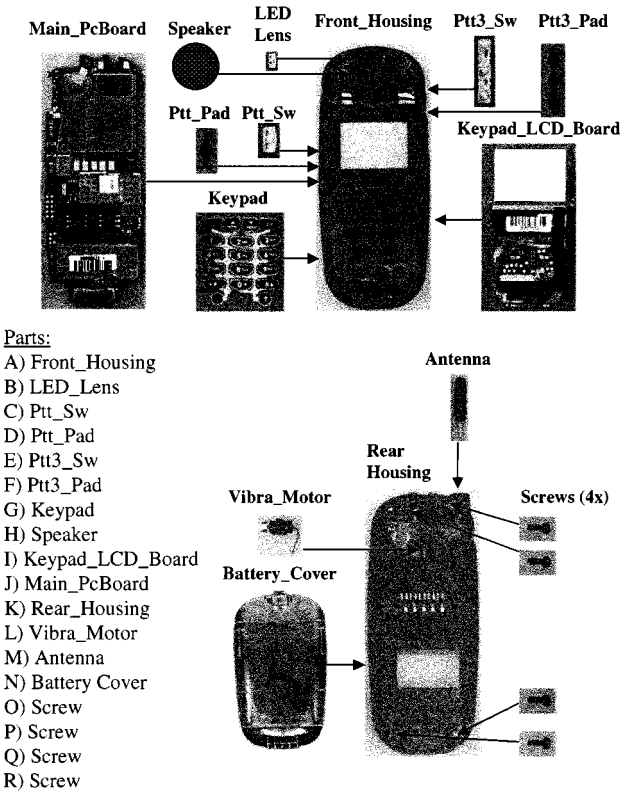

Fig. 4. Telephone with 18 parts used in the simulation.

IV. Assembly and Disassembly Cost

From the generated assembly and disassembly tree, the total assembly and disassembly cost can be formulated as follows: total cost $=$ operation cost $+$ handler cost.

The operation cost is equal to the sum of all costs associated with SACS operations. Handler cost is the total cost of all handlers required for performing the assembly or disassembly operations. For a product that is fully reversible in terms of assembly and disassembly operations, the total cost is obtained by adding all operations as we traverse the assembly and disassembly tree from its leaves to the root. Similarly, total disassembly cost is obtained by traversing the same tree from the root to its leaf nodes. However, if a product that cannot be disassembled by simply applying the assembly operations in reverse or by using the same set of handlers, a different disassembly tree should be generated.

In practice, cost increases with the increased capability of tools; thus, tool selection can significantly influence the final assembly and disassembly cost. In a prototyping environment, handlers' costs will dominate the total cost. Conversely, operation cost dominates in production since it will increase linearly with the number of products made, while handlers' costs are effectively lowered as they get amortized over a larger number of product units. Maximizing handler reuse is always beneficial, especially in low-volume production. In high-volume production, design should also focus on minimizing operation cost such as cycle time. At times, it may even make sense to use a higher-cost handler to reduce its operation cost. In the long run, an optimized solution may include a set of standardized GENAD workcells. Then, most parts will be handled by the standard workcells while some custom handlers are used for the remaining (small) set of unique parts.

V. Simulation Results

Simulations were carried out on a telephone handset. The product consists of 18 primitive parts, as shown in Fig. 4. Parts and handlers required for assembly and disassembly operations were modeled using the described GENAD workcell methodology. Each operation for merging and separating parts was assigned a SACS code, as shown in Figs. 5 –7.

To simplify our simulation explanation, the following abbreviations were used. First, each part is given an alphabetic name: A through R. As before, parts and handlers are denoted as P and H, respectively. Secondly, the operators named OP1 and OP2 are set to "$+$" and "$-$", or to "$-$" and "$+$", for assembling or disassembling the product, respectively. Finally, lines connecting parts and handlers in the assembly tree are coded using two types of line, thicker-line and thin-line, representing the stationary part and moving part, respectively, during assembly or disassembly operation.

As part of our simulation, we used the designer's telephone assembly tree to generate a detailed assembly and disassembly tree by following the GENAD workcell guidelines. Two key parts named A and K are important as they hold all the remaining parts in place. Part A accepts parts B through J,

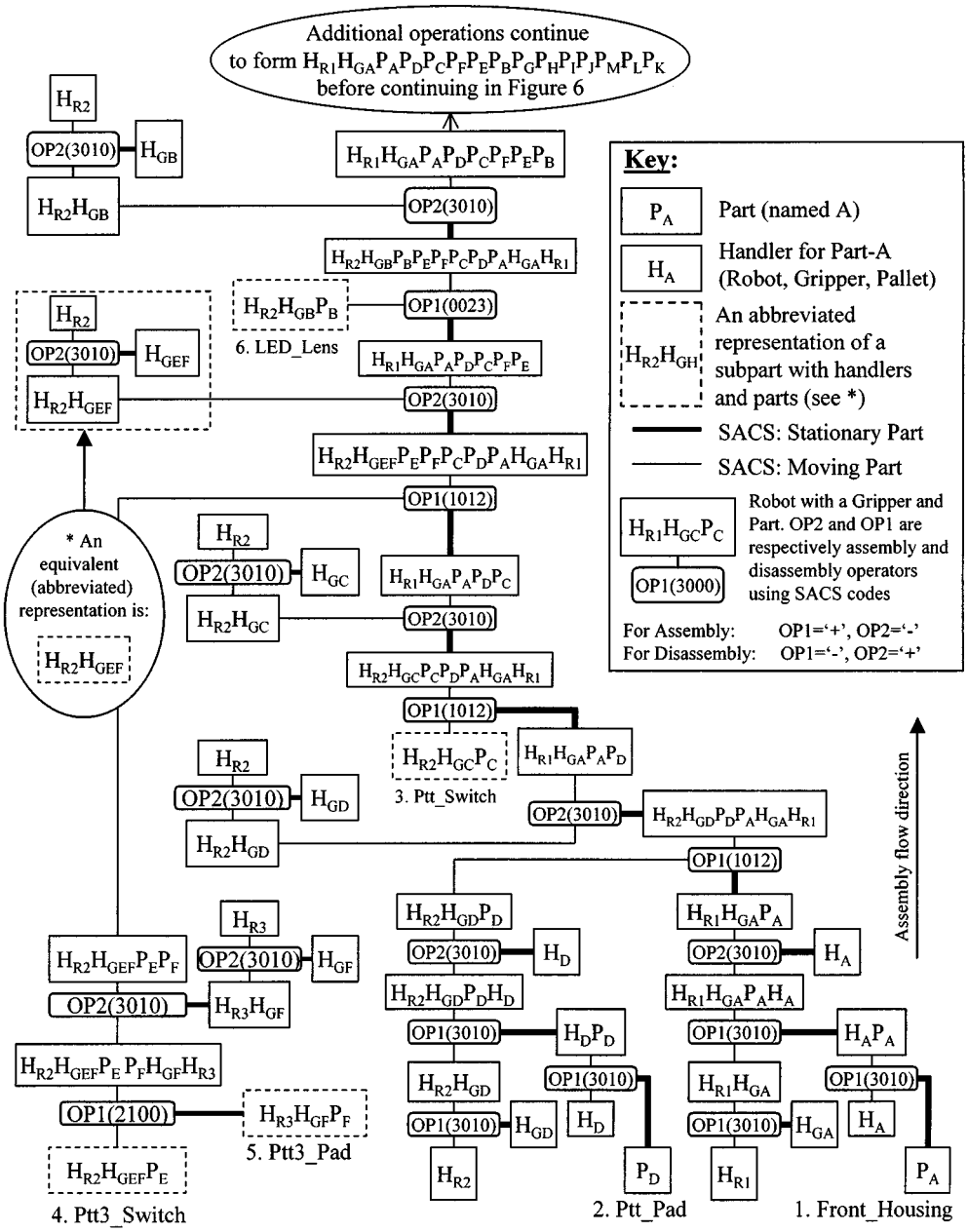

Fig. 5. Assembly and disassembly tree generated for the telephone simulation.

while part K accepts parts L through R. Four screws named $O, P, Q,$ and R hold the two key parts A and K together after assembly.

We will now trace their operations as described in the generated assembly and disassembly tree, as shown in Figs. 5–7. The assembly process starts with P_A (Front_Housing) and P_D (Ptt_Pad), in Fig. 5. They are initially loaded into their respective feeders named H_{FA} and H_{FD}. Once loaded using operator $\{+(3010)\}$, the resulting "feeder A with part A" and "feeder D with part D" are denoted "H_AP_A" and "H_DP_D," respectively. Part grippers for P_A and P_D (abbreviated as H_{GA} and H_{GD}) are similarly attached to robots one and two (abbreviated as H_{R1} and H_{R2}) to form $H_{R1}H_{GA}$ and $H_{R2}H_{GD}$, as illustrated in the lower part of Fig. 5. To assemble parts A and D, $H_{R1}H_{GA}$ moves to the location of H_AP_A. When part P_A is grasped by $H_{R1}H_{GA}$, the two objects will merge to become a single object labeled $H_{R1}H_{GA}P_AH_A$. Before the robot can remove P_A from feeder H_A, a disassembly operation $\{-(3010)\}$ separates H_A from $H_{R1}H_{GA}P_A$. The handler HA is now an empty feeder.

A second robot H_{R2} performs similar operations on H_{GD}, H_D, and P_D to form $H_{R2}H_{GD}P_D$. It then moves to attach itself to $H_{R1}H_{GA}P_A$, forming $H_{R2}H_{GD}P_DP_AH_{GA}H_{R1}$. Robot $H_{R2}H_{GD}$ proceeds to release P_D that is now attached to $H_{R1}H_{GA}P_AP_D$. If robot $H_{R1}H_{GA}$ also releases Part P_A at this point, we would have an assembled part called P_DP_A. However, in Fig. 5, the assembly operation continues by attaching $H_{R1}H_{GA}P_AP_D$ to $H_{R2}H_{GC}P_C$ to form $H_{R2}H_{GC}P_CP_DP_AH_{GA}H_{R1}$. (Notice that robot H_{R2} is being re-used for part P_C after assembling P_D.) When $H_{R2}H_{GC}$

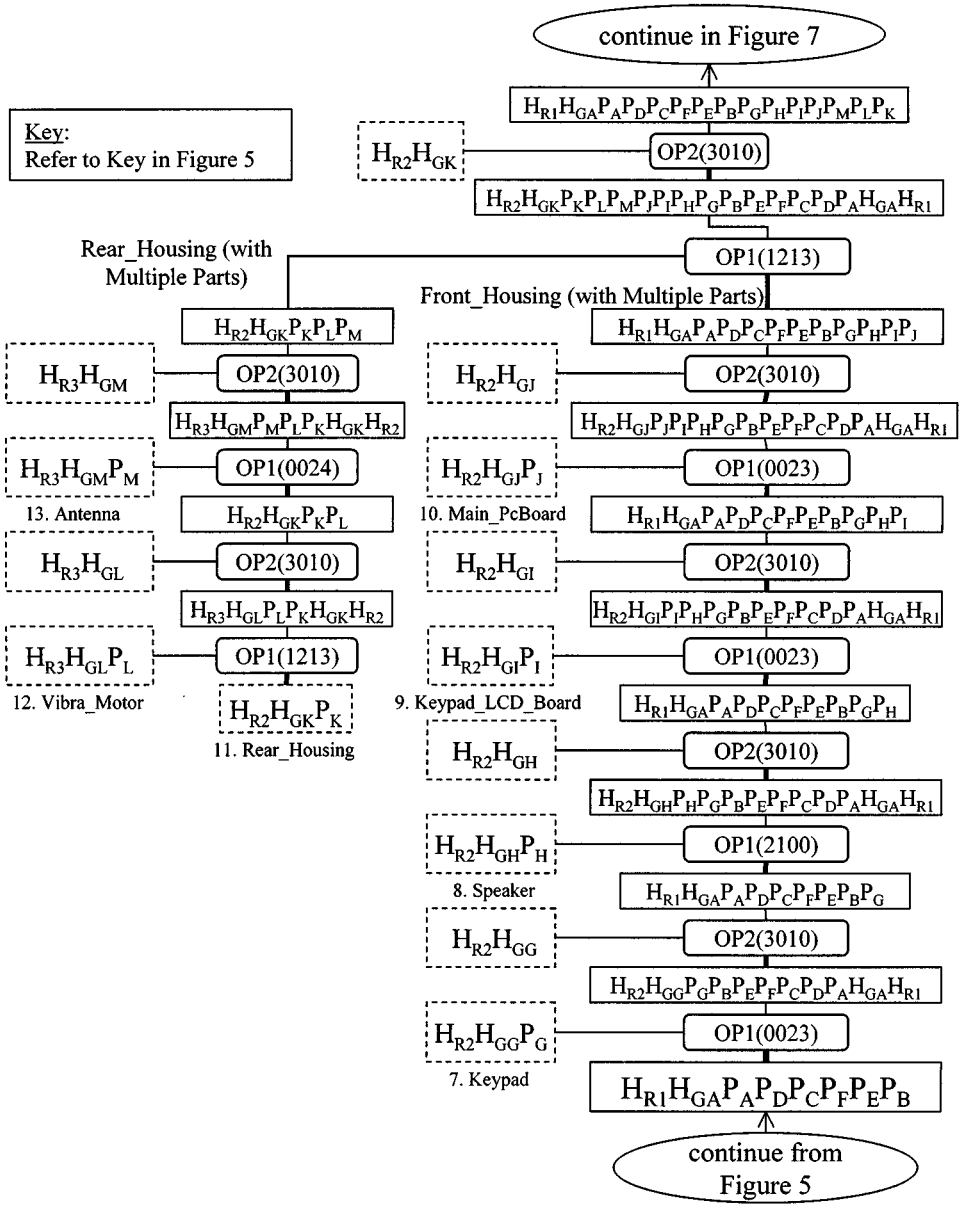

Fig. 6. Generated telephone assembly and disassembly tree continued from Fig. 5.

releases P_C, the three parts $P_AP_DP_C$ are now attached to robot one, represented by $H_{R1}H_{GA}P_AP_DP_C$. The remaining operations are executed in a similar manner. To reduce clutter in the figures, some detailed operations were not shown explicitly. For example, the LED_Lens ($H_{R2}H_{GB}P_B$, inside a dotted box) near the top of Fig. 5 has a set of associated part fetching operations that are not shown.

In Fig. 6, four more parts are assembled into the front housing to form $P_AP_DP_CP_FP_EP_BP_GP_HP_IP_J$, grasped by $H_{R1}H_{GA}$. In addition, the antenna and Vibra_Motor are assembled into the back housing to form $P_KP_LP_M$, grasped by $H_{R2}H_{GK}$. The back housing operations could have been executed in parallel to the front housing to reduce cycle time if more handlers were used. Subsequently, to hold the front and back housings together, a custom handler named H_{SOPQR} drives four screws (P_O, P_P, P_Q, and P_R) into the assembled parts. The handler can load four screws simultaneously but it dispenses them one at a time, as shown in the middle of Fig. 7. As a result, the tree has four assembly and disassembly operations using the same handler. Finally, a Battery_Cover (P_N) is assembled to the telephone. The completed telephone is represented by $P_NP_RP_QP_PP_OP_KP_LP_MP_JP_IP_HP_GP_BP_EP_FP_CP_DP_A$.

For disassembling the telephone, the disassembly tree can be interpreted in a similar fashion. However, the operation starts from a fully assembled product, at the top of the assembly tree. In addition, the operators OP1 and OP2 are now set to "$-$" and "$+$," respectively. Initially, an assembly operation creates $H_{R1}H_{GA}$ (robot number one and its gripper). Then, a second set of handlers $H_{R2}H_{GN}$ attaches to the telephone body $(P_AP_D \ldots P_QP_R)$ and the battery cover (P_N). A disassembly operation $\{-(0023)\}$ then separates P_N from the main telephone body $(P_AP_D \ldots P_QP_R)$. Following that, P_N is placed onto H_N, which is now more appropriately referred to as a recycling bin for P_N.

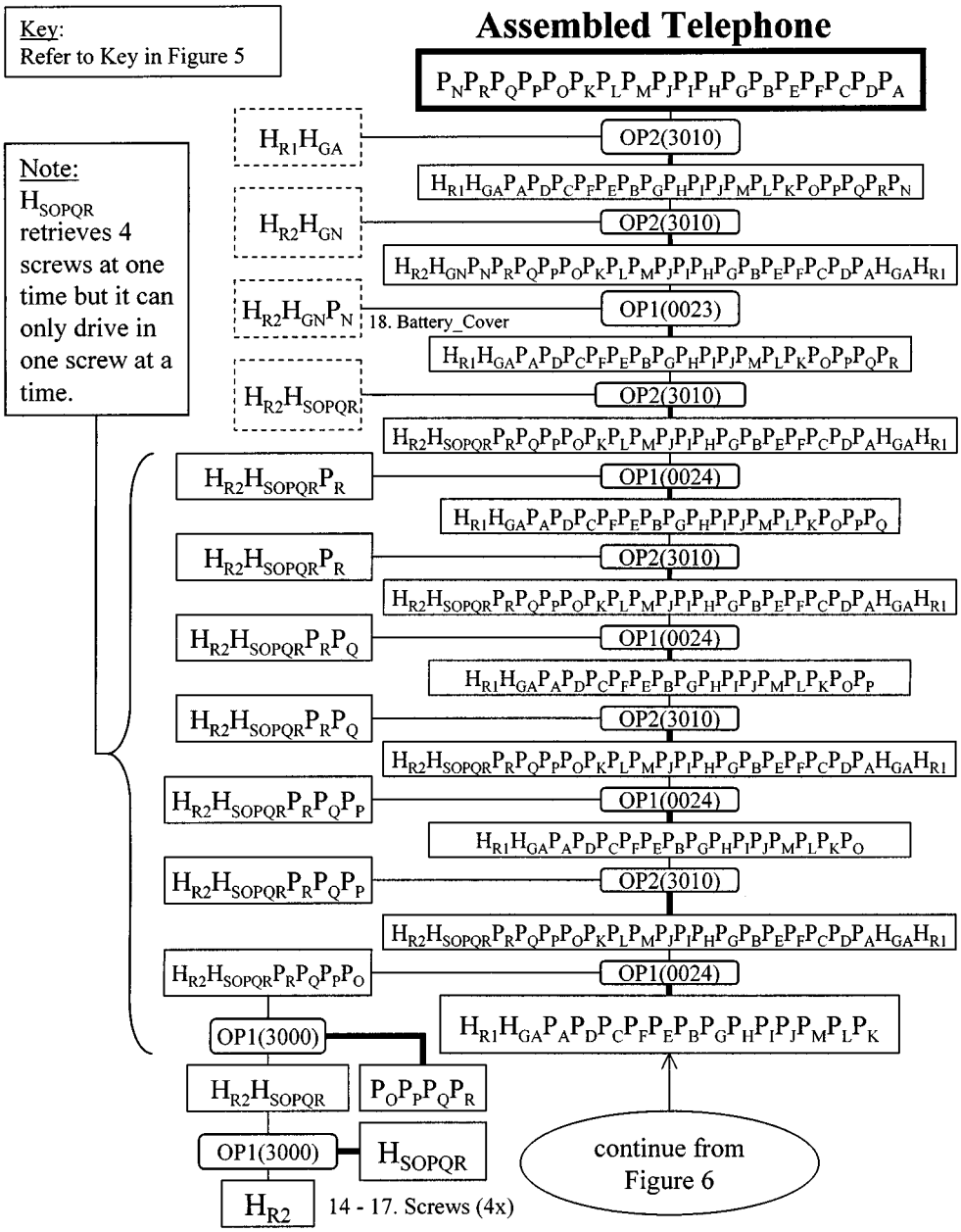

Fig. 7. Final part of telephone assembly and disassembly tree ends from Fig. 6.

In this simulation, all parts were considered fully recyclable and they could be disassembled using the same assembly handlers. However, most real products have parts that are not designed for disassembly. In such cases, the assembly and disassembly trees will be different.

In the assembly and disassembly trees, each SACS code is associated with two operation costs related to assembly and disassembly, respectively, shown in Fig. 8 By summing up these individual costs for all SACS operations in a assembly tree, a total cost for product assembly and disassembly can be attained. Based on the SACS table defined in Fig. 8, the estimated maximum assembly operation cost for assembling the simulated telephone is equal to $(47^*c3010)+(2^*c2100)+(3^*c1012)+(4^*c0023)+(2^*c1213)+(5^*c0024)+(2^*c3000)+(42^*d3010)$. In this example, SACS operation 3010 was used 47 times for assembling parts and was used 42 times for disassembling parts. In addition, the cost for the required handlers will be $H_A+H_D+H_C+H_F+H_E+H_B+H_G+H_H+H_I+H_J+H_M+H_L+H_K+H_{SOPQR}+H_N+H_{GA}+H_{GD}+H_{GC}+H_{GEF}+H_{GF}+H_{GB}+H_{GG}+H_{GH}+H_{GI}+H_{GJ}+H_{GM}+H_{GL}+H_{GK}+H_{GN}+H_{R1}+H_{R2}+H_{R3}$. When OP1 and OP2 are reversed, each assembly operation becomes a disassembly operation and vice-versa. In the simulation, every handler was assumed to be capable of performing both assembly and disassembly operations; thus, the cost of disassembling this telephone can be evaluated by $(47^*d3010)+(2^*d2100)+(3^*d1012)+(4^*d0023)+(2^*d1213)+(5^*d0024)+(2^*d3000)+(42^*c3010)$. In this case, SACS operation 3010 was used to disassemble parts 47 times and to assemble parts 42 times. In most practical situations, the product assembly and disassembly operations are not

						Cost	
Level 0	*Level 1*	*Level 2*	*Level 3*	*Part Mating Type*	*Code*	*Assembly*	*DisAssem.*
0 (Hole)	0R	P	Z	against-fit (null)	0023	c0023	d0023
		P	W	screw-fit (null)	0024	c0024	d0024
	1R	P	Z	fit (Rz)	0123	c0123	d0123
		V	Y	side-fit (Ry)	0112	c0112	d0112
	2R	V	Y	e-fit (RxRz)	0212	c0212	d0212
		P	Z	e-side-fit (RxRy)	0223	c0223	d0223
	3R	V	Z	sph-fit (RxRyRz)	0313	c0313	d0313
1 (Groove)	0R	V	Y	semi-against-fit (Ty)	1012	c1012	d1012
	1R	V	Z	semi-fit (TyRz)	1113	c1113	d1113
		P	Y	semi-side-fit (TyRy)	1122	c1122	d1122
		H	X	side-semi-fit (TyRx)	1131	c1131	d1131
	2R	V	Z	e-semi-fit (TyRxRz)	1213	c1213	d1213
		O	Y	e-semi-side-fit (TyRxRy)	1222	c1222	d1222
		H	X	e-side-semi-fit (TyRyRz)	1231	c1231	d1231
	3R	V	Z	sph-semi-fit (TyRxRyRz)	1313	c1313	d1313
2 (Plane)	1R	N	N	against (TxTyRz)	2100	c2100	d2100
	2R	N	N	e-against (TxTyRyRz)	2200	c2200	d2200
	3R	N	N	sph-against (TxTyRzRyRx)	2300	c2300	d2300
3 (Handler)	0R	C	N	stationary-compliant (null)	3000	c3000	d3000
		P	N	stationary-position (null)	3010	c3010	d3010
	1R	C	N	1axis-compliant (null)	3100	c3100	d3100
		P	N	1axis-position (null)	3110	c3110	d3110
	6R	C	N	6axis-compliant (null)	3600	c3600	d3600
		P	N	6axis-position (null)	3610	c3610	d3610
	XR	C	N	Xaxis-compliant (null)	3X00	c3X00	d3X00
		P	N	Xaxis-position (null)	3X10	c3X10	d3X10
Custom	(New custom SACS codes can be added later to handle new operations)						

Key: Type 0,1,2,3: V=Vertical P=Parallel H=Horizontal N=Don't Care
Type 3: Level 1: Degree-Of-Freedom Level 2: C=Compliant P=Position
X=0,1,2,3,…,n degree of freedom

Fig. 8. Assigned SACS codes from the developed structured assembly coding system.

so easily interchangeable. Moreover, handlers are usually designed to only perform either assembly or disassembly operations. As a result, their assembly and disassembly operation cost equation will not be the same.

VI. Conclusion

A system for assisting product designers to recognize and correct potential manufacturing problems in their product is presented in this paper. The proposed virtual assembly and disassembly (VIRAD) system can be integrated into a CAD/CAM environment. It is set up for encouraging product design intervention during the design process so as to prevent badly designed parts from ever reaching production. The developed system uses a model of generic assembly and disassembly (GENAD) workcell, which in turn uses a developed structured assembly coding system (SACS) to describe assembly and disassembly operations.

To illustrate the developed system, a telephone assembly and disassembly simulation was carried out. The results showed that actual assembly and disassembly operations could be modeled successfully. High-level product assembly instruction is first obtained from a CAD system. A binary assembly and disassembly tree is then generated. Each part is assigned with a set of handlers for assembly and disassembly operations. In our model, we use string notation to manipulate parts and handlers for assembly and disassembly. Furthermore, based on cost assigned to each SACS operation, the total cost for assembling or disassembling a product can be obtained by summing up the costs required for all SACS operations and handlers.

In our future work, an algorithm for generating optimized GENAD assembly and disassembly trees will be developed. In addition, automatic task planning for assembling and disassembling parts based on SACS-defined operations will also be studied. In addition, a muscle-like compliant-control [8], [9] will be implemented on a robot to execute these low-level SACS operations.

APPENDIX

The previously developed structured assembly coding system (SACS) [5], [6] can encode all mating-operations for assembling two primitive parts in a three-dimensional (3-D) task space. It uses a force compliant system to define assembly operations [8] between a *stationary* part and a *moving* part. The geometric constraints between two mating parts, a moving part and a stationary part, are used to determine the assembly or disassembly process. The basic SACS coding system uses four levels (the first four columns in Fig. 8) to code 24 types of part-mating (the fifth column in Fig. 8) operations defined for primitive parts. The corresponding SACS codes for these 24 types are listed in the sixth column of Fig. 8. SACS is a generic coding system based on the geometrical constraints between two parts. A detailed description of how to define SACS codes can be found in [5].

For the purpose of developing a model of GENAD workcell in this paper, a new set of class-3 objects called handlers, such as fixtures, robots, end-effectors, pallets, feeders are also defined by the SACS code. For example, the two handlers with SACS codes of 3000 and 3010 are grippers with and without force compliance control, respectively. A Level-0 code of 3 (the first left digit of the SACS code) is always assigned to handlers. Level-1, the second digit of the code, describes the number of degrees-of-freedom (DOF) the handler possesses. A handler with 6R (a Level-1 code of 6) is a robot with six DOF, while a 1R (a Level-1 code of 1) device can be a part conveyor. Level-2 code uses "0" and "1" to represent "C" and "P," corresponding to force compliance and position (no force compliance) control, respectively. The Level-3 code is not currently defined for this group. It can potentially be used for describing special properties such as machine vision fiducial marks.

Finally, each SACS code is associated with a product assembly and disassembly cost, as shown in columns 7 and 8 of Fig. 8. Its cost is an object containing a set of information, such as cycle time, tooling cost, and assembly yields.

ACKNOWLEDGMENT

The authors wish to thank I. Turlik, T. Babin, and M. Serrar of Motorola Inc. for their support in virtual manufacturing.

REFERENCES

[1] H. Srinivasan, N. Shyamsundar, and G. Rajit, "A virtual tool to support environmentally conscious product design," in *Proc. IEEE Int. Symp. Electron. Environment*, San Francisco, CA, May 5, 1997, pp. 7–12.

[2] B. H. Lee and I. Kosuke, "Demanufacturing complexity metrics in design for recyclability," in *IEEE Int. Symp. on Electron. Environment*, San Francisco, CA, May 5, 1997, pp. 19–24.

[3] S. Mok, C. H. Wu, and D. T. Lee, "A system for automatic assembly and disassembly operations," in *Proc. IEEE Int. Conf. Robotics Automat.*, vol. 4, San Francisco, CA, Apr. 24–28, 2000, pp. 3695–3700.

[4] ——, "A hierarchical workcell model for intelligent assembly and disassembly," in *Proc. IEEE Int. Symp. Comput. Intell. Robotics Automat.*, Monterey, CA, Nov. 8–9, 1999, pp. 125–130.

[5] C. H. Wu and M. G. Kim, "Modeling of part-mating strategies for automating assembly operations for robots," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 1065–1074, July 1994.

[6] M. G. Kim and C. H. Wu, "A formal part mating model for generating compliance control strategies of assembly operations," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Los Angeles, CA, Nov. 1990, pp. 611–616.

[7] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers, Principles, Techniques, and Tools*. Reading, MA: Addison-Wesley, 1986.

[8] C. H. Wu, "Compliance control of a robot manipulator based on joint torque servo," *Int. J. Robot. Res.*, vol. 4–3, pp. 55–71, 1985.

[9] C. H. Wu, K. S. Hwang, and S. L. Chang, "Analysis and implementation of a neuromuscular-like control for robotic compliance," *IEEE Trans. Contr. Syst. Tecnol.*, vol. 5, pp. 586–597, Nov. 1997.

Swee M. Mok (M'95) received the B.Sc.(Hon.) degree in aeronautical and astronautical engineering from Southampton University, Southampton, U.K., in 1985 and the M.S. degree in aerospace engineering from the University of Cincinnati, Cincinnati, OH, in 1987. He is currently pursuing the Ph.D. degree in electrical and computer engineering at Northwestern University, Evanston, IL.

He is a Senior Staff Engineer at Motorola Labs, Schaumburg, IL. He joined Motorola in 1987, where he worked on two-way radio manufacturing. Some of his recent work includes the development of a production process for attaching fine-pitch flip chips onto organic substrates, high throughput surface mount technology equipment, and wireless biomedical sensors. His current research interests include CAD/CAM, assembly and disassembly of electronics products using intelligent force-compliant robots, and embedded devices for control and communication.

Chi-haur Wu received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, R.O.C. in 1973 and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, in 1981.

He has been with Department of Electrical and Computer Engineering at Northwestern University, Evanston, IL, since 1983. Before joining Northwestern, he was with Unimation Robotics Inc., Danbury, CT. During that period, his job involved designing robot motion control algorithms and digital servo systems for PUMA robots and hydraulic-servo Unimate robots. His research interests include engineering design and invention, which includes robotics, human augmentation control, active suspension control, active force absorbing control, vibration isolation, microcontroller and embedded systems, electronic circuit design, sensor-circuit design and control, robotic motion planning and control, compliance and damper control, pattern recognition, integrated industrial automation, automated assembly/disassembly systems, CAD/CAM industrial automation, computer-assisted surgical robot systems, rehabilitation robots, medical images and instrumentations, and remote telemanipulation through Internet and RF.

D. T. Lee (S'75–M'78–SM'84–F'92) received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, R.O.C. in 1971 and the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana-Champaign, in 1976 and 1978, respectively.

Prior to joining the Institute of Information Science, where he is a Distinguished Research Fellow and Director, he was a Professor of the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL, where he has worked since 1978. From 1989 to 1990, he was the Program Director for the Computer and Computation Theory Program with Division of Computer and Computation Research of the National Science Foundation. His research interests include design and analysis of algorithms, computational geometry, VLSI layout, parallel and distributed computing, web-based computing, algorithm visualization, software tools development, compliant controller for active suspension and vibration control, digital libraries, and advanced IT for intelligent transportation systems. He has published over 120 technical articles in scientific journals and conferences. He is Editor of *Algorithmica, Computational Geometry: Theory and Applications, ACM Journal of Experimental Algorithmics*, Chief Editor of the *International Journal of Computational Geometry and Applications*, Editor-in-Chief of the *Journal of Information Science and Engineering*

Dr. Lee is a fellow of ACM and a member of IICM.