

Roy S. Freedman and Robert P. Frail

# OPGEN: The Evolution of an Expert System for Process Planning

## Definition of the Problem Domain

*Process planning* is an activity that is performed routinely by industrial engineers in order to schedule and allocate resources for equipment assembly. The planning activity consists of the following: Given a collection of electrical component parts of a printed circuit board together with its layout diagram, derive the sequence of operations required of a technician (or robot) for the correct assembly of the printed circuit board. The plans that result are called *operations sheets*.

At Hazeltine, process planning is an industrial engineering activity that occurs after design and before assembly. In the initial stage of process planning, the industrial engineer carefully examines a parts list (obtained from the bill of materials file; see figure 1) and a detailed physical layout diagram (see figure 2), that shows the proper placement of the corresponding parts on the printed circuit board. The bill of materials files are located on a Hazeltine IBM business computer. The physical layout diagrams are prepared at a Hazeltine computer-aided design (CAD) facility. After studying these data, the industrial engineer applies expertise and experience to produce a detailed plan that specifies (a) which components are to be installed; (b) the factory work center of the installation procedure; (c) details associated with the installation procedure, such as special tooling or special intermediate manufacturing activities or processes, and (d) when the installation is to be performed. This detailed plan is written by the industrial engineer on operations sheets. These operations sheets are then used by production assemblers to perform the actual printed circuit board assembly. The assembly steps typically involve both the manual insertion and the machine insertion of components (utilizing special computer-aided manufacturing equipment).

---

Roy S. Freedman is an associate professor at Polytechnic University in the Department of Electrical Engineering and Computer Science, Farmingdale, New York 11735. He also serves as a consultant for Hazeltine Corporation, Greenlawn, New York 11740, as well as other organizations. Robert P. Frail is a software engineer in the research laboratory at Hazeltine Corporation, Greenlawn, New York 11740 and a senior teaching fellow at Polytechnic University where he is pursuing his Ph.D.

## Previous Approaches with Conventional Technology

In late 1981, operations sheets were handwritten by industrial engineers (see figure 3). The creation of these operations sheets was an informal, tedious exercise. Previous approaches to the problem of automating the generation of operations sheets was considered in conventional technology by the business data processing department that serviced the industrial engineering groups. The data processing department regarded this task as infeasible for several reasons.

(1) there was very little documentation that described the actual process-planning task, (2) the informal guidelines used by the industrial engineers to generate operations sheets were always being revised in accordance with specific assembly runs, and (3) the industrial engineers required a high degree of interactive capability with the system in order to accommodate the various "exceptional conditions" that arose during small assembly runs of nonstandard printed circuit boards. The data processing department was used to automating tasks that were well understood, well documented, and formally specified by system analysts. As is discussed later, the automation of process planning became an ideal candidate for a knowledge-based expert system approach.

What makes the OPGEN task different from other similar configuration tasks is the magnitude of possible component configurations: An industrial engineer can encounter almost 50 thousand different parts in this planning domain. In practice, when an industrial engineer encounters an

---

**Abstract** The operations sheets generator (OPGEN) is an expert system that helps industrial engineers at the Hazeltine manufacturing and operations facilities plan the assembly of printed circuit boards. In this article, we describe the evolution of OPGEN from its initial development in the Hazeltine research laboratories to its routine use in an integrated manufacturing environment. We describe our approaches to the problems that occurred during the development, integration, and rehosting of OPGEN and provide some methodological guidelines to expert system builders who are concerned with the final delivery of an expert system.

---

```

PART NO: C5872968          CKT CD ASSY,FDBK IF AMP          DATE: 02 28 25
E C NO:                    COMP NO:                    ITEM:                    PC:
QUANTITY EF START          PC: OPTS/VRNTS
LN ITEM COMPONENT PART NO.  PER ASSY.  ADD REV  DEL REV  CRY UNL A  B
01 0013 C1804E208J80P      EA  1.00000      P
CAPACITOR
01 0014 C1804E338J80P      EA  3.00000      P
CAPACITOR
01 0015 C1804E528J80P      EA  1.00000      P
CAPACITOR
01 0016 C1804E328J80P      EA  1.00000      P
CAPACITOR
01 0017 C135A200           EA  1.00000      P
CAP VAR CERAMIC
01 0018 2N471N4148-1      EA  3.00000      P
DIODE
TO CONTINUE SCANNING, PRESS ENTER

```

Figure 1. *OPGEN* Input: Unparsed Parts List from Bill of Materials Database.

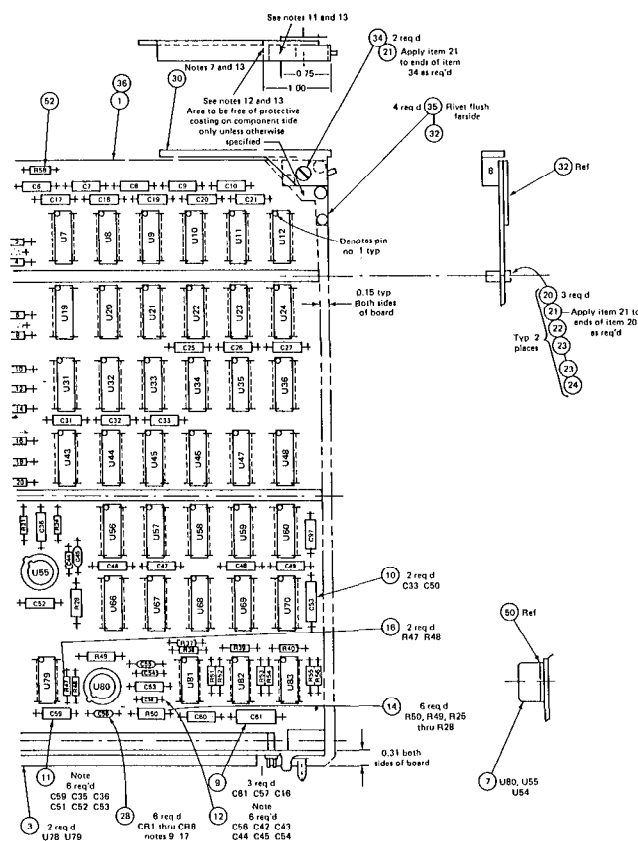


Figure 2 *OPGEN Input Circuit Layout Diagram.*

unknown part, typically a set of questions are asked of a knowledgeable source (a colleague or a manual) in order to acquire this new part installation knowledge. OPGEN emulates the industrial engineer's ability to acquire this knowledge.


 <b>ASSEMBLY OPERATION PROCESS</b> INDUSTRIAL ENGINEERING DIVISION			
ASS'Y NO. <b>124323</b>		DESCRIPTION <b>LF CONTROL BOARD ASSY</b>	
OPER. NO.		REVISION DATE	
WIP CTR.	ITEM NO.	OPERATION DESCRIPTION	
40	921	HAND EXAMINE THE FOLLOWING OPERATIONAL POINTS	
		A3 REL FIGURE # _____	
	122	1	S3L77 CONNECTOR W/70°
	118	2	MS-2047A23-4 PINET
	53	10	R50196-2 TEST POINT

Figure 3 Example of a Handwritten Operations Sheet

## Initial Development Approach

In the following eight subsections, we present a brief discussion of methodology for expert system development, selection of problem and tools, knowledge engineering and prototype implementation, operational feasibility, and the actual development of a working prototype of a process planning expert system.

## Methodology for Expert System Development

Expert systems require a software development methodology that differs in some respects from those methodologies used for conventional systems. Most knowledge-based development methodologies used by organizations experienced in building expert systems are similar in that they concentrate on the early (feasibility) stages of a project. Very little has been published on the later stages, which are concerned with expert system delivery, integration, and maintenance. During the development of OPGEN, we incorporated the lessons learned in these early stages and revised our original approach to provide for integration and maintenance.

Most expert system development methodologies are a variation on the following theme, which paraphrases Hayes-Roth (1985): (1) expert system technology is determined to be relevant to a product; (2) management provides an opportunity for action; (3) a preliminary business application is assembled; (4) a knowledge engineering consultant verifies the opportunity; (5) a knowledge engineering project team is formed and assesses the knowledge; (6) the knowledge engineering project manager plans the project; (7) the user organization supports the plan; (8) project management and coordination are established; (9) the knowledge engineering project team starts on the project; (10) periodic reviews and reports track project progress and expectations; (11) pilot testing occurs; (12) project enhancement and delivery requirements are defined; (13) project control transfers from the knowledge engineering project team to the users; (14) the project is assimilated into the organization as a result of

acceptance, reliability, and confidence; (15) development is completed, and maintenance begins; and (16) new requirements and opportunities surface.

Steps 1-5 are concerned with application selection; steps 6-12 deal with establishing the operational feasibility of the expert system by building an operational specification (a prototype). Steps 8-12 encompass the knowledge engineering tasks of the system. Steps 13-14 are concerned with integrating the expert system into existing hardware and software systems and into existing organizational practices. Steps 15-16 handle rehosting, transporting, and maintaining the final deliverable.

All of these steps were performed during the development of OPGEN. Steps 1-12 were performed initially in 1982 and are discussed in Frail and Freedman (1984) and Rauch-Hindin (1985). Steps 13-14 were completed last year and are discussed in Freedman and Sylvester (1985). Steps 15-16 are discussed in Frail and Freedman (1986).

### **Selection of the Problem and Tools**

In late 1981, the Hazeltine Technical Planning Department and the research laboratories were looking for a mechanism to incorporate the new advances in knowledge-based technology into Hazeltine products. We decided that the best way to learn how to incorporate this technology was to develop a prototype system which would solve some practical problem. Our threefold approach was (1) to find the right kind of problem, (2) to find the right kind of knowledge engineering tools, and (3) to deliver the prototype expert system. This approach corresponds to steps 1-12 in the methodology.

There were several requirements for a suitable problem-solving domain. We were looking for a problem domain in which there were several available experts who could solve a particular problem in a few person-days using known procedures (not necessarily formal) and experience. We did not want our expert system to completely replace the human expert because we considered the development of such a system infeasible. In addition, the results of such an expert system might not be trusted by the human experts. We required that our system be a job performance aid that would increase the productivity of the human expert and would provide the human expert with the ability to modify the conclusions of the expert system. Ideally, the problem should be regarded as time consuming and tedious by experts in order to assure maximum user acceptance. We selected the domain of process planning for the assembly of printed circuit boards. Our experts would be industrial engineers.

Our next step was the selection of tools. In early 1982, we believed that our process-planning expert system would be based on rule-based productions, like several other planning expert systems that were known at the time (McDermott 1980). UCI-Lisp (an InterLisp subset) and MacLisp were already installed on our DEC-10. We obtained the MacLisp version of the OPS5 production system interpreter from

Charles Forgy (1981) at Carnegie-Mellon University (CMU).

The MacLisp version of OPS5 had some very important features that we exploited later. This version allowed us to define our own functions and right-hand-side actions. It also provided important interfaces to the MacLisp environment. This flexibility allowed us to extend our basic tool set.

### **Knowledge Engineering and Prototype Implementation**

We performed in-depth interviews with the industrial engineering experts over a period of several months. The purpose of these interviews was to determine what rules the experts used in writing operations sheets. These rules would then be represented in OPS5. At this time, the difference between this knowledge engineering activity and the system analysis associated with conventional software engineering practice became apparent to the experts and knowledge engineers. The industrial engineers were accustomed to explaining the requirements of an activity and expected a system analyst to formulate a system specification that solved a problem. They were not accustomed to constantly explaining how they perform their activity and what "rules" they used. This knowledge engineering activity, essentially concerned with representing the cognition of the industrial engineers in a knowledge-based formalism (such as productions), continually surfaced exceptions to previously elicited rules. Sometimes, the industrial engineers were reluctant to admit their ad hoc process-planning decisions. From the knowledge engineering perspective, we were concerned that the only way to fully understand the process-planning problem was to train the knowledge engineer in industrial engineering or, vice versa, to train the industrial engineer in Lisp-OPS5 programming.

The initial implementation of the expert system also posed some practical problems regarding its use and interaction with other computer systems. The bill of materials portion of the input had to be obtained through a batch process in which an IBM operator read the appropriate magnetic tape. This tape was then given to a DEC-10 operator who wrote the tape to a DEC-10 file. The file then had to be converted from an EBCDIC format to ASCII format. This procedure was necessary because no direct link between the business and scientific computers was available at our installation.

Other significant problems concerned the representation of the geometric information on the layout diagram. We determined that the only information in these diagrams relevant to the preparation of operations sheets was the indication of simultaneous part installations. On the diagram, these installations were shown as a series of interconnected circles. For example, if the physical layout diagram showed item numbers 16, 17, and 23 linked together, these parts were to be installed as a set. The knowledge engineers believed that a file containing these part sets could be generated

easily by an industrial engineer; the industrial engineer need only be taught how to record these sets in a file (the *sets file*) of a preestablished format using a text editor.

Another problem was the lack of consistent operations sheet format conventions. Prior to this expert system effort, each industrial engineer had developed a personal style for formatting operations sheets. This personalization created a lack of format consistency. The implementation of this expert system forced them to specify uniform formatting conventions for their operations sheets. These conventions were very important because the operations sheets were ultimately read by assembly production people.

### **Operational Feasibility**

In spite of the knowledge elicitation and system interface problems, we felt that operational feasibility could be established. Productions were encoded from the consistent and well-defined planning rules, and in November 1982, a working prototype of the process-planning expert system was implemented and reviewed. It was called the Hazeltine artificial intelligence laboratory-1 (HAIL-1) and contained approximately 80 OPS5 productions and several pages of external Lisp functions and user-defined actions.

HAIL-1 productions encoded the installation sequence of these parts using the "procedure-by-context" method (described in detail later) to transfer flow control to different sets of productions (a production set). Each production set, averaging about six rules, was responsible for writing installation instructions for a single part name (for example, "TRANSISTOR"). Included in each set were productions to check for the existence of part instances in the input and to transfer flow control to the next production set when no instances were left.

### **How HAIL-1 Works**

HAIL-1 performed its process-planning task by reading the bill of materials file and sets file input into its working memory and then transferring control to the first part installation production set. Like other planning systems, HAIL-1 used the weak problem-solving method of match (as implemented in the OPS5 interpreter) as its search strategy. Here, any part instances in working memory matching a production in this production set were removed from working memory after installation instructions were written. Control was then transferred to the next production set in the part assembly sequence, which performed similarly. HAIL-1 continued in this way until all production sets were "visited." In the case where the input contained no instances of a particular part, the corresponding production set simply transferred control to the next production set in the sequence without writing any instructions. Although the HAIL-1 part installation sequence was "hard wired" (that is, could not be changed under program control), this state was of no consequence because the part installation sequence had a high degree of temporal stability.

HAIL-1 contained eight production sets (about 50 productions) that could plan the installation of about 40 parts. (This number of parts is greater than the number of production sets because different part instances often share the same name. For example, there are several part instances named "RESISTOR.") The remaining productions performed user interface functions.

HAIL-1 was not a trivial system and was regarded by the knowledge engineers as a demonstration of operational feasibility. However, although the knowledge engineers considered HAIL-1 usable, the industrial engineers did not. The most important issue concerned the way that the bill of materials file was acquired. HAIL-1 users found this process awkward, frustrating, and time consuming. Some other issues included the revision of the operation descriptions, the addition of page numbers (with form feeds selected adaptively to improve readability), and the generation of a work summary file describing all the operations for a particular assembly.

Some of the errors uncovered included the incorrect sequencing of operations and the incorrect determination of which parts were machine insertable and which were hand insertable. The industrial engineers also insisted that the HAIL-1 knowledge base was inadequate for the job: More knowledge about different parts and processes had to be provided.

The completion of the HAIL-1 prototype also raised certain management issues for the future. In December 1982, HAIL-1 was a research vehicle under development by the research laboratories for use by manufacturing and operations. If HAIL-1 was to become a product, who would own it, and who would maintain it? In addition, who would make the decision to adopt HAIL-1 as an industrial engineering tool.

### **OPGEN: Expert System as Product**

In early 1983, HAIL-1 was not considered a finished product. In order for HAIL-1 to be regarded as a significant job performance aid, we realized that the following improvements needed to be implemented. The short-term improvement was to expand the knowledge base. HAIL-1 had the process-planning knowledge that enabled it to write operations sheets for a small fraction of the total number of electronic parts which could be encountered in a typical printed circuit board assembly. Additional knowledge engineering was required to generate more rules to adequately cover additional parts and processes in the problem-solving domain.

The intermediate-term improvement was to integrate HAIL-1 into other systems and practices. The system had to be more fully integrated with the other computer systems that made up the Hazeltine manufacturing control system in order to improve the access to the bill of materials files.

The long-term improvement was to extend the problem-solving domain. The industrial engineers wanted HAIL-1 to be extended to other process-planning activities involving

operations sheets. These extensions could be horizontal (extensions to similar process-planning tasks, such as printed circuit fabrication, cable harnessing, and hybrid fabrication) or vertical (extensions to more complex activities such as the automated programming of a machine that inserts integrated circuits).

The problem of increasing the knowledge base was regarded as crucial. However, it became apparent that even at an estimated break-even point of one thousand productions (at which point our time-shared DEC-10 would become completely unusable), there still would be insufficient knowledge about all the different types of components. We also noted that very few industrial engineers have the installation knowledge of the approximately 50,000 different parts. In a case where an industrial engineer encounters a part for which no personal installation knowledge exists, the engineer either consults with other experts or tries to find the required information in the part manufacturer's catalog. In order to increase the HAIL-1 knowledge base, we concluded that the best approach was to simulate how an industrial engineer learns about new part installation procedures and to automate this knowledge-acquisition mechanism within HAIL-1.

Our first attempt at automatic part installation knowledge acquisition was based on a metarule approach in which productions create and modify other production sets. Based on the assumption that on the order of 1000 production sets (at six productions per part) are required to represent the installation knowledge associated with 50,000 parts, the implementation would have to be capable of handling at least 6000 productions, a number well beyond our break-even point (Execution time would also suffer because these additional capabilities would have to be controlled by the OPS5 interpreter, these capabilities could not be compiled.) A greater technical problem was the implementation of a group of productions that created new production sets. Synthesizing the proper antecedents and consequents under program control turned out to be very cumbersome in OPS5. For these reasons, we rejected the metarule approach as infeasible.

Because of the infeasibility of implementing the metarule approach, to automate knowledge acquisition, we developed an efficient approach during fall 1983. We separated the domain knowledge into productions and frames (schemas). We decided to use the productions to specify general planning knowledge associated with process planning and to use frames to specify detailed operations specific to a part or process (this knowledge could be represented hierarchically) The resulting system, renamed OPGEN, was demonstrated in December 1983.

## How OPGEN Works

As we added more production sets to HAIL-1, we noticed they were nearly identical except for certain information

about the specific part with which the production set was associated. We generalized the production set by factoring out the part-specific knowledge associated with each production set and representing this knowledge in a way that was easily accessed and modified. These last characteristics were required for our proposed approach to an automated knowledge-acquisition scheme. The representation we chose to satisfy these requirements was frames.

Production memory was abstracted into two generic production sets: one for single-part installation (5 productions) and one for set installation (about 10 productions). This abstraction also eliminated knowledge of the part assembly sequence from these productions. Because this knowledge was also part-specific, it was thus represented in the part-specific frame base.

The revised system also read the input files into working memory, as HAIL-1 did. However, each part, upon being read, was assigned a time tag that indicated its relative order in the part assembly sequence. This time tag was obtained by using each part name as an index into an associative memory (the \*PART → TIME\* frame in the part-specific knowledge base). A time-tagged working memory element encapsulating all the part-specific information was then created for each part; for example:

```
(part ^Item 13 ^Name RESISTOR ^Prt RC707 ^Qty 2 ^Time 4).
```

To complete the system, control productions were added to implement an internal "clock" loop. This loop counted up from time one to the last installation time and applied the generic part installation productions to the part-specific working memory elements at each time step. In this way, installation instructions for those parts with time tag t were written at time t. (A partial production antecedent for implementing this match is shown later.)

The most significant aspect of this hybrid production-frame architecture was that it allowed us to implement an efficient automated knowledge-acquisition mechanism. This mechanism was triggered when a part's time tag was nil, indicating that it wasn't instantiated in the part-specific knowledge base. Control was then transferred to the OPGEN knowledge-acquisition dialogue to permit the user to instantiate a new slot in the \*PART → TIME\* frame.

This version of OPGEN writes operations sheets by firing productions that are sequenced by accessing instantiated frames. In operation, OPGEN first parses the bill of materials file (see figure 1) and the sets file obtained from the layout diagram (see figure 2). OPGEN then associates the information in the sets file and bill of materials file (for a given printed circuit board assembly) with OPS5 working memory elements. The part-specific knowledge contained in the frames is also established in working memory. When OPGEN encounters a part with an empty frame slot (an unknown part), it asks the user to fill in a new slot. The OPGEN knowledge-acquisition dialogue solicits the information nec-

essary to create a new slot with the new part installation knowledge.

Figures 4-7 depict portions of this dialogue. In figure 4, OPGEN asks the user to identify the correct time slot for the unknown part. Figure 5 shows OPGEN soliciting a description of how to install the unknown part (the operation description). This description gets instantiated as a slot in one of the OPGEN frames.

If new knowledge is acquired during a process-planning session, OPGEN allows the user to choose either permanent retention or transient retention of this knowledge. *Permanent retention* refers to the retention of the newly acquired knowledge for this and all subsequent planning sessions. *Transient retention* indicates that the newly acquired knowledge is retained for this planning session only. This flexibility is important because of the dynamic nature of the knowledge associated with our process-planning task domain. This aspect of the dialogue is shown in figure 6.

Figure 7 shows an example of an OPGEN-generated operations sheet. After writing operations sheets, OPGEN writes a special page listing those parts for which it still has no part installation knowledge. This listing is issued if the user declines to tell OPGEN about an unknown part. OPGEN also writes the work summary of all operations specified on the generated operations sheets.

## OPGEN: Expert System as a Component of an Integrated Manufacturing System

By early 1984, most of the OPGEN knowledge engineering problems were solved. We developed a user operations manual (describing the preparation of OPGEN inputs and the teaching of OPGEN about new installation procedures) and a systems manual (describing how the OPGEN executable files can be recreated from the source code files). These manuals were written for users who are inexperienced in computer programming or in DEC-10 operations.

Other integration tasks were also performed. The interface to the IBM computer was reconfigured. The bill of materials file was now obtained using a personal computer as an intermediary: The file would be written from the IBM computer to a floppy disk on the personal computer, translated from EBCDIC to ASCII, and then written to the DEC-10. This procedure was also somewhat awkward but was much faster than the tape-transfer procedure.

OPGEN was used on an experimental basis starting in July 1984. Most complaints about OPGEN during its initial use were not concerned with its process planning. The two biggest complaints concerned the computer interfaces and operations sheet editing. Because OPGEN was run from the manufacturing facility (situated about 40 miles from the main plant), the reliability of data transmission between computers was dependent on the availability of good communications links. A noisy link would corrupt the input file and frustrate the OPGEN users.

```
*****) I have no knowledge of INT-CKT-PWR-GATE,
p/n 911264, item 6. Do you wish to instruct me
about this now, Rob?
(Strike Y for Yes and any other key for No) } y
```

```
Strike the Space Bar when ready for more information
Do you wish to see the part insertion time menu?
(Strike Y for Yes and any other key for No) } y
```

```
Scan the following operation sequence for the number that
represents what a/an INT-CKT-PWR-GATE is either
installed with or installed after:
```

```
(0 (PCB PC-BOARD BOARD-PRINTED-WIRING-
BOARD)
(1 (PRECISION-OP-AMPQUAD-NAND-GATE
N7-400J-MICRO-CIRCUIT MICRO-CKT MICRO-CIRCUIT
INTEGRATED-CKT)
(2 (MACHINE-INSERTABLE)
(3 (TAB PIVOT INSULATOR CONNECTOR CONNECTOR-
ASSY BRACKET BRACKET-ANGLE-&-SPACER
BRACKET-PIVOT)
```

```
Strike the Space Bar when ready for more information  ☐
```

Figure 4. OPGEN Interactive Knowledge-Acquisition. Identification of Unknown Part.

```
Indicate by the appropriate number (followed by a carriage
return) what a/an INT-CKT-PWR-GATE is either installed
with or installed after} 11
```

```
Confirm: (11 (WAVE-SOLDER ACCORDANCE WITH
MIL-STD-454 Requirement 5) ) correct?
(Strike Y for Yes and any other key for No) } y
```

```
Is a/an INT-CKT-PWR-GATE installed either With (strike Y)
or installed After (strike any other key) this part group or
operation:
(WAVE-SOLDER ACCORDANCE WITH MIL-STD-454
Requirement 5) } n
```

```
Confirm: After?
(Strike Y for Yes and any other key for No) } y
```

```
Do you wish to create an Operation Description for
this part?
(Strike Y for Yes and any other key for No) } y
```

```
Enter the Operation Description for INT-CKT-PWR-GATE,
p/n 911264, item 6, then strike Return:
} Please install the power gate with Sue's Tweezers.
```

```
Confirm: Should the Operation Description say:
Please install the power gate with Sue's Tweezers.
(Strike Y for Yes and any other key for No) } ☐
```

Figure 5. OPGEN Interactive Knowledge-Acquisition. Filling in Frame Slots.

The second complaint concerned "fine tuning" the operations sheets generated by OPGEN. We believed that any additional modifications to the operations sheets could be

```
*****) I have no knowledge of INT-CKT-FLIP-FLOP, p/n
910704-2, item 35.
Do you wish to instruct me about this now, Rob?
(Strike Y for Yes and any other key for No) } n
```

Still examining the parts.....

```
Do you wish to permanently record this new information
(I'll save the old information too) we've discussed so far,
Rob?
(Strike Y for Yes and any other key for No) } n
```

I'm re-examining the parts with this new information in mind. This may take as long as sixty seconds, Rob.....

Please enter the Assembly Number, then Return } 123190

Now Please enter the Assembly Description, then Return  
} My Easy OPGEN Demo Board

I will now generate op sheets.....

```
--> Time 1 < - > Quit time 24
--> Time 2 < - > Quit time 24
10: Load Axial Lead Components ☐
```

Figure 6. OPGEN Alternatives to Knowledge Retention.

ASSEMBLY NO: 124323  
DESCRIPTION: RF Control Board Assembly

OPER. NO.	WK. CTR. NO.	ITEM NO.	QTY.	OPERATION DESCRIPTION
(cont.)	50	902		Hand install the following TRANSISTORS as per figure #
	14	1		2N2219 TRANSISTOR (note the polarity) with:
	60	2		the AN960C3 WASHER-FLAT
	49	2		the MS35338-134 WASHER-LOCK
	59	2		the MS51957-3 SCREW
	73	1		the 340354 HEAT-SINK
	14	5		2N2219 TRANSISTOR (note the polarity) with:
	17	5		the 670076 PAD
	13	4		2N2222 TRANSISTOR (note the polarity) with:
	16	4		the 670099 PAD
	63	8		2N2369A TRANSISTOR (note the polarity) with:
	16	8		the 670099 PAD

Time 1 < - > Quit time 24

Figure 7 Example of Operations Sheet Generated by OPGEN

accomplished easily by editing the OPGEN outputs. However, if the OPGEN users were unfamiliar with the resident screen editor, this editing operation was also time consuming and tedious. This problem was solved by creating command procedures that would easily manipulate the operations sheet data.

## The Current Approach to OPGEN Development

In the following six subsections, we present a brief discussion of the rehosting of OPGEN and the recoding of OPGEN's productions, the new OPGEN knowledge base architecture, the implementation of OPGEN in Common Lisp and how it works, and finally an evaluation of approaches to OPGEN development.

## The Rehosting of OPGEN

The availability of sophisticated development environments enables iterative prototyping of production-based systems. The high degree of interaction they provide permits a system prototyping capability which is much more rapid than that permitted by any conventional procedure-based, compiled-language development environment. This rapid prototyping capability is needed for demonstrating system feasibility.

However, in 1984 OPGEN was in the delivery stage of expert system development. Its feasibility had been proven sufficiently by its production-based ancestor, HAIL-1. In addition, its viability as a prototype had been demonstrated by its industrial engineering users over a 12-month period; it had been in everyday use as a commercial product within a Hazeltine manufacturing facility for 15 months.

As a result of this experience, the OPGEN production base was well understood: It was thoroughly debugged and remained of fixed size over time, independent of any knowledge acquired by its frame base. (The OPGEN frame base more than doubled in size as its users directly provided new part-specific knowledge, without any intervention from knowledge engineers.) Consequently, the OPGEN production system representation had outlived its usefulness because its planning task was well understood, and an operational specification had been developed. Most of OPGEN was already coded in MacLisp. Because of these factors and the potential increase in maintainability and transportability, we undertook the task of recoding the OPGEN production system rule base in Common Lisp. Our rehosting goal was to keep the OPGEN artificial intelligence-based knowledge representations and problem-solving methods and remove the artificial intelligence (AI) development tool (OPS5). In practice, this rehosting was accomplished without any change to the OPGEN inputs, outputs, and knowledge-acquisition dialogue (see figures 4-7).

Rehosting was also necessary for integrating OPGEN into other factory automation facilities. As observed in Freedman and Sylvester (1985), Fox (1983), and Frail and Freedman (1984), the integration of an expert system within the techno-cultural milieu of the user is important for the realization of any projected cost reduction.

As part of a factory integration plan, OPGEN was integrated into the Hazeltine manufacturing control system (Freedman and Sylvester 1985) in November 1985. OPGEN was rehosted from its DEC-10/TOPS-10 environment to a VAX/VMS environment. This was done as part of an exper-

imental factory automation program—the material storage, transfer, entry, and routing system (MASTERS)—sponsored by the U. S. Air Force. The goal of the MASTERS program is to automate the manufacturing processes of selected Air Force contractors in order to reduce the cost of Air Force procurements.

### The Recoding of OPGEN Productions

The DEC-10 version of OPGEN employed the MacLisp-based OPS5 production system to represent general knowledge of printed circuit assembly and frames to represent part-specific knowledge. The productions contained many calls to supplementary MacLisp functions. For example, the following production excerpt from the OPGEN rule base contains a call to the Lisp function TIME-OF:

```
(P ASSIGN-TIME-TAG
  (context-assign-time-tag)
  (part ^Item <i> ^Name <n> ^Prt <p> ^Time nil)
  - (set <g> set-item <i> set-qty <q>)
  →
  (remove 1)
  (make context-unknown-part?)
  (modify 2 ^Time ($value (TIME-OF <n> <p>))))).
```

The final version of OPGEN had a rule base of 62 rules that made 31 calls to external Lisp routines (including both functions and user-defined actions) and had a frame base which grew to contain knowledge of 115 parts. One problem that surfaced during the rehosting was maintaining the communication between OPS5 and Lisp.

During tests of Common Lisp and OPS5 for VAX, we discovered that communication between the two was possible only through an intermediate language (for example, FORTRAN), the VMS mail utility, or some combination of the two. We rejected this approach when we read in the "VAX Lisp User's Guide" (1984) that "programs written in other VAX languages cannot call VAX Lisp routines." DEC confirmed that this meant OPS5. The VMS mailbox approach was rejected because it was too system dependent.

Another proposed solution to this communication problem was to bypass the issue altogether and rewrite the OPS5 rules in Lisp, making the system entirely Lisp based. This would eliminate the inefficiency of running interpreted OPS5 (OPS5 for DEC-10 can't be compiled; OPS5 for VAX can) and increase OPGEN transportability to other Common Lisp installations. Coincidentally, this task had previously been considered for the DEC-10 version of OPGEN in order to reduce OPGEN execution time.

### The New OPGEN Knowledge Base Architecture

Many existing industrial rule-based expert systems use rules exclusively to represent all types of knowledge. OPGEN knowledge was partitioned into separate representations, as described earlier, in order to implement an efficient knowledge-acquisition process. Because new rules are diffi-

cult to generate under program control, we factored OPGEN dynamic, part-specific knowledge out of its rule base and represented it in a way most amenable to the required modifications. The result was a system of hybrid architecture whose part-specific knowledge base grew as it acquired new manufacturing knowledge but whose rule base size was independent of any part-specific knowledge and thus remained constant over time. The decision to recode the OPGEN rule base in Lisp was made easier by the distributed architecture of the knowledge base.

This decision was also made easier by the fact that during any execution of OPGEN, the flow of control within its rule base passed through groupings of functionally related rules in a fixed sequence. The production-based engineering technique that we call procedure by context is used to enforce a certain flow of control through the productions. This technique is also common to many rule-based expert system development tools (for example, "salience" in ART [Rauch-Hindin 1985]). A similar technique is also discussed in Brownston et al (1985). In the simplest implementation of the technique, each production is given an antecedent that must match the current "context," where a context is a special element in working memory that denotes some step in a procedure (for example, "install electrical components"). Each grouping of productions related to the same step specifies the same context antecedent. Every production in a grouping which could possibly be the last to fire within that grouping is responsible for changing the current context to the next context in the sequence. (A failure to change contexts is often the cause of infinite loops in production systems.) The production shown earlier illustrates the use of the technique to move from one context (context-assign-time-tag) to another (context-unknown-part?). In this way, rule-based procedures are implemented. For example, the OPGEN steps (contexts) for installing sets of parts are roughly (1) read sets, (2) identify set leaders, (3) time tag set leaders, (4) install a set leader, and (5) install remainder of set.

Associated with each of these contexts was an average of four productions. Some production groupings were written such that the action denoted by the context name could be iterated as required, thus forming a local loop. In addition, the procedure-by-context technique was used to iterate on a more global level over contexts four and five combined, as required.

In spite of the procedural nature of most rule-based systems, rules are fast becoming the knowledge representation of the masses (Chandrasekaran 1985). Production systems are generally more suitable for exploratory and iterative prototyping than for procedure-based development systems. This is in part due to the well-publicized modular nature of productions as "independent pieces of knowledge" that can be added easily to the knowledge base (Barr and Feigenbaum 1983). However, this independence is overrated when one considers that every production added usually has to be



“spliced” into its proper context. Bugs in production systems are often caused by productions firing “out of context.” In a system of more than a few hundred productions, knowledge maintenance becomes increasingly difficult because of the relative loss of modularity. As Georgeff (1982) observed, this technique obscures the representation of flow-control information in production systems, causing a decrease in potential extensibility and maintainability.

### The Implementation of OPGEN in Common Lisp

OPGEN productions exhibited a well-defined procedural nature despite the fact that its control flow was obscured by its distribution among production antecedents and working memory. Because of this, the representation of OPGEN flow-control knowledge in a procedural representation was fairly straightforward. The most interesting aspect of this task was representing the OPS5 working memory elements of production-based OPGEN in Common Lisp. Global variables and constants were used to represent unstructured data, such as the current time, the current operation number, and the user’s name. For structured data, we implemented a representation called *transient frames*, frames, (in the form of a-lists) that are dynamically created, used, and then discarded during a run of OPGEN. For example, part information in the OPS5 working memory was represented as follows, one for each part:

```
(PART ^Item <i> ^Name <n> ^Number <no> ^Quantity <q> ^Time <t>).
```

Such a representation, combined with the pattern matching performed during the recognize-act cycle of the OPS5 interpreter, provided an easy way to access part attributes using any other part attribute as the key. For example, to find the item numbers and quantities of those parts whose insertion time is the current time, the antecedent of a production specified the following condition elements:

```
(PART ^Item <i> ^Quantity <q> ^Time <t>)
(CURRENT-TIME <t>).
```

To find the names and numbers of those parts having different insertion times and different item numbers, one could specify:

```
(PART ^Name <n> ^Number <no> ^Time <t1> ^Item <i1>)
(PART ^Name <n> ^Number <no>
  ^Time {<t2> <> <t1>} ^Item {<i2> <> <i1> }).
```

These examples show the access requirements of the data structures within OPGEN productions. These access requirements were satisfied in Common Lisp with the use of transient frames. For example, to find the item numbers and quantities of those parts whose insertion time is the current time, OPGEN (after reading the parts file) first creates the transient frame \*TIME → ITEM-QTY\*, which maps part insertion times to part item numbers and part quantities. This frame is created using the permanent frame \*PART → TIME\* (in the part-specific knowledge base referred to ear-

lier) to obtain a part’s insertion time. The current time would then be used as a search key into the \*TIME → ITEM-QTY\* frame to collect the item numbers and quantities of those parts whose insertion time is the current time. Similar transient frames of the a-list type were used to represent relations between other part and insertion time attributes. The entity-relationship model of data (Ullman 1980) was used to design the particular frame schemes that satisfied OPGEN Common Lisp-based data-access requirements. We considered implementing transient frames with structures as an alternative to a-lists. However, structures were rejected for two reasons: (1) associative memories provided more appropriate and efficient data access and (2) the overhead from the creation of the slot constructing, slot accessing and slot assignment macros associated with the structures was greater than the overhead incurred in creating the a-lists.

### How the Current OPGEN Implementation Works

The Common Lisp-based version of OPGEN performs the printed circuit board assembly planning task in exactly the same way that production-based OPGEN performs the task. Its inputs and outputs are identical (notwithstanding extensions to the new version). All parts are time tagged upon input. Unknown parts trigger the same knowledge-acquisition dialogue. (The knowledge-acquisition dialogue required no changes because the part-specific knowledge base that it modifies remained unchanged.) Its clock loop (implemented in terms of OPS5 productions in the previous version) was reimplemented in Lisp. As in the production system version, match was employed by the clock to determine the installation time of part instances.

The only significant difference between the two versions is the large gain in system maintainability. This gain resulted from the appropriate representation choice for the procedural knowledge.

### An Evaluation of the Approaches to OPGEN Development

The VAX Lisp-based version of OPGEN was installed on a VAX/11-750 in March 1986 after three person-months of work. This version is 100% compiled Common Lisp. Two-thirds of the DEC-10 version consisted of MacLisp forms that implemented general utilities and frame-manipulation functions (these were compiled); the remaining one-third of the system consisted of OPS5 productions that implemented the general planning productions. These productions were written in 882 lines of OPS5; it was rewritten in 441 lines of Common Lisp. Table 1 compares various parameters of three versions of OPGEN. When taken together, these three versions span three distinct evolutionary stages of expert system development (feasibility, prototype, and commercial product).

HAIL-1 represents the system in its feasibility stage. OPGEN-P (for production based) represents the system in

Table 1. Comparison of Different Implementations of OPGEN.

	HAIL-1	OPGEN-P	OPGEN-L
Stage	Feasibility	Prototype/Product	Product
Period	11/82-9/83	9/83-3/86	3/86-present
Rules	86	62	0
% Rules	80	33	0
% LISP	20	67	100
Parts Known	30	44-115	115
Knowledge Acquisition			
Ability?	no	yes	yes
Run Time (sec)	120	90	45

both the prototype and commercial product stages. OPGEN-L (for Common Lisp based) represents the system in the commercial product stage.

Run times indicate the approximate time for a particular version to generate operations sheets for a test file of 44 known components (consisting of 28 distinct parts). No knowledge acquisition was necessary. As the system progressed to a totally Lisp-based implementation, the run time decreased. This decrease must be attributed to the noncompilability of the DEC-10 OPS5 productions rather than to the production-based representation itself.

### The AI in OPGEN

In the following three subsections we present a brief discussion of the pragmatics and the AI aspects of OPGEN knowledge representation, and new observations in expert system development methodology.

#### Pragmatics

OPGEN evolved from an implementation dependent on a knowledge engineering tool to an implementation independent of tools. The architecture of its knowledge base also evolved from a rule-based production system representation to a multiple representation consisting of productions and frames to a procedure- and frame-based representation (implemented entirely in Common Lisp). OPGEN has been in use for planning the assembly of over 90% of the new printed circuit boards at Hazeltine and will be used for a major production run during the manufacture of the Hazeltine microwave landing system for the Federal Aviation Administration. Figure 8 shows a diagram of the OPGEN current knowledge flow.

#### The AI Aspects of OPGEN Knowledge Representation

The decomposition of OPGEN knowledge into a hybrid organization that separates general planning knowledge and part-specific knowledge has several benefits. One of the most significant benefits stems from the observation that the general planning knowledge base can be applied to any plan-

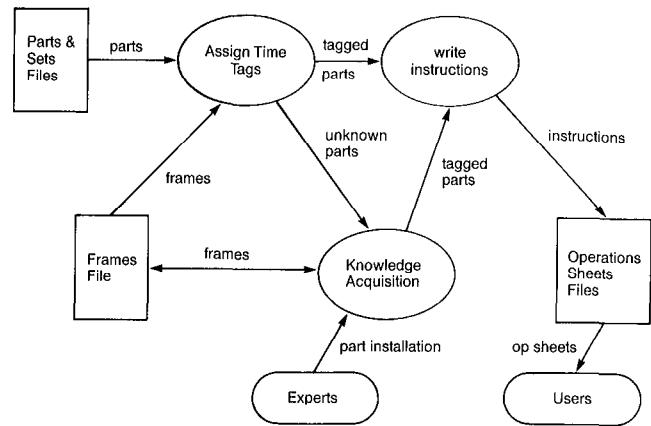


Figure 8. OPGEN Knowledge Flow.

ning process for which an operation-specific knowledge base is constructed. For example, an expert system for hybrid circuit fabrication can be configured by conjoining the general planning knowledge base with an operation-specific knowledge base containing knowledge of the operations involved in the fabrication of hybrid circuits. In this sense, OPGEN can be considered a generic process-planning system.

Another major AI aspect of OPGEN modular knowledge representations pertains to knowledge acquisition. By separating the operation-specific knowledge from the general planning knowledge base (and representing it in a way that is more amenable to modification), we implemented an efficient mechanism for acquiring operation-specific knowledge directly from the OPGEN user. This knowledge-acquisition mechanism is executed under the control of the general planning knowledge base.

Given the OPGEN knowledge-acquisition mechanism, it is possible for OPGEN to learn new part installation knowledge "from scratch." This ability can be demonstrated by applying the general planning knowledge base to a *tabula rasa* (Locke 1690), an empty part-specific knowledge base. This is the extreme case in which all parts on a parts list are unknown.

#### Expert System Development Methodology: Some New Observations

Our experience in developing OPGEN led to the following observations and guidelines concerned with delivering knowledge-based systems. If knowledge-based systems are engineered with these guidelines in mind, we believe that expert system life-cycle management will be easier.

**Observation 1:** Delivery of an expert system should be considered in the earliest stage of development.

**Guideline:** The decision about when prototyping, feasibility, and knowledge engineering end and where system delivery and maintenance begin is the most important and per-

haps the most difficult step in the evolution of the expert system product (Polit 1985). A robust knowledge architecture should be specified as early as possible in order to address the issues involved in integrating and rehosting a final deliverable.

**Observation 2:** Proper knowledge elicitation requires the utmost tact and diplomacy because defensive reactions and feelings of inadequacy can potentially be invoked in the experts.

**Guideline:** Several factors create tension and defensiveness during knowledge elicitation interviews of experts. Perhaps the greatest factor is the fear that they will be replaced by a machine. We avoided this issue by choosing to automate a task that the experts were glad to relinquish. Another factor concerns the fear of inadequacy felt by the assigned experts. The process of knowledge elicitation usually forces the experts to think about their jobs at a previously unconsidered level of detail. This can create feelings of inadequacy that invoke defensive or less than candid responses.

We recommend not referring to experts as experts. Such a label implies a responsibility that makes many people feel uncomfortable. It tends to create (in the assigned experts) the expectation that they must know all answers immediately. In reality, some expert knowledge might be hard to recall because it is seldom used or might simply be unknown to a particular expert. One might argue that such people are not real experts. During the evolution of OPGEN, we were fortunate enough to have experts who were usually certain of their knowledge—even when it conflicted with their colleagues' knowledge.

**Observation 3:** Production rules are useful for rapidly prototyping a small system for the purpose of demonstrating feasibility.

**Guideline:** After the feasibility stage, productions are useful for representing static knowledge; other representations should be considered at this point for the representation of dynamic knowledge. A proper design should emphasize the functional decomposition of knowledge and then the choice of appropriate representations. Representation choice should also be influenced by requirements of interoperability, codability, modifiability, understandability, maintainability, and transportability.

**Observation 4:** Adding productions does not necessarily improve an expert system.

**Guideline:** Our experience with OPGEN taught us that productions are a poor way to represent knowledge which is intended to grow. Yet, the pervasive impression is that the more productions, the better. In fact, the constant need to update a production base often indicates a severe mismatch between the knowledge representation choice and the knowledge representation task. The increase in domain knowledge as a result of addition to a production base is accompanied by a disproportionate decrease in system understandability and maintainability. The acquisition of new domain knowledge

by humans usually improves their problem-solving ability; this is not yet universally true for rule-based expert systems.

In addition, although production-based development tools sometimes provide a metarule capability, production-based knowledge is not generally context-free, as we pointed out earlier. As we discovered with HAIL-1, even in the simplest cases there is much overhead involved in determining what context a potential new production should fire in, what contexts it should make and remove, what other condition elements are required, and what other actions should be taken. Similar guidelines (based on the XCON experience) are found in Van de Brug, Bachant, and McDermott (1985).

**Observation 5:** If an expert system product has successfully been used to solve problems in several separate but similar domains, then it might be cost effective to transform that system into a shell.

**Guideline:** OPGEN contains general planning knowledge of sequences associated with the generation of operations sheets. It is possible to generalize it to sequence any type of entity associated with the generation of operations sheets, given the appropriate task-specific knowledge base. Such an operations sheet planning shell, in conjunction with a task-specific knowledge base editing facility, would permit users to create their own modular task-specific knowledge bases. The users of such systems would then become solely responsible for their knowledge base content.

This task is currently being considered in the case of OPGEN. In the degenerate case where the OPGEN part-specific knowledge base is empty, all parts in an input file are flagged as unknown (the case of the *tabula rasa* discussed earlier). In this situation, the industrial engineer is able to "bootstrap" the system by directly providing all installation knowledge.

**Observation 6:** These questions might arise: Is the final deliverable system still an expert system? Is it still AI?

**Guideline:** From the perspective of the person implementing the system, to say it no longer is AI is to say either that the designation depends upon the implementation or that the implementation makes the system. If this were true, we could write an OPS5 program to compute factorials and call it an expert factorial system. Moreover, an expert system can be implemented with any tool: As long as the expert system exploits explicit AI knowledge representations and problem-solving methods, it should be regarded as an application of AI technology. Even though OPGEN is not currently implemented in terms of AI tools, it is still based on the AI knowledge representation of frames and still utilizes match as a problem-solving method. In some respects, this issue, which Chandrasekaran (1985) calls "the computational universality of representation languages" (such as OPS5 or other expert system building tools) typically causes expert systems to "lose perspicuity of representation."

From the perspective of the user, it is irrelevant to the users (at system delivery time) whether their problem-solving system was prototyped using an expert system devel-

opment tool. However, their system might never have been implemented (or delivered on time) if an operational specification was not demonstrated by knowledge engineers using expert system building tools or AI knowledge representations and problem-solving methods.

### The Status of OPGEN Today

In these final three subsections we will present a brief discussion of the current status of OPGEN, including the use of OPGEN to help solve a new automatic programming problem and the OPGEN integration scheme.

### A Vertical Extension to OPGEN

As the feasibility of knowledge-based systems is demonstrated, extensibility of these systems becomes important. OPGEN was extended in the last year to generate the instructions needed to program a dual in-line package (DIP) insertion machine. This machine automatically inserts DIPs into sockets on printed circuit boards. Prior to this extension, OPGEN listed on an operations sheet those parts which were insertable by this machine.

### The New Problem: The Automatic Programming of the DIP Insertion

One of Hazeltine's manufacturing facilities houses an Amistar C1-1800 automatic DIP insertion machine (DIM). This machine must be programmed to insert a sequence of DIPs into a corresponding sequence of socket locations on a printed circuit board. Because different boards contain different DIPs and DIP locations, the machine must be programmed for each different board that it processes. DIM is normally programmed by an industrial engineer. In order to subsume this programming task, OPGEN was programmed to emulate the activities of the industrial engineer. The industrial engineer uses a layout drawing of the printed circuit board under construction to generate a DIP insertion program. From this drawing and from other knowledge, the engineer determines the sequence of DIP insertion locations and the corresponding DIP input sequence to the machine. These two items constitute the DIP insertion program and are respectively documented by the industrial engineer as the programming sheet and the chute loading sheet.

The layout drawing provides necessary but insufficient programming information. The "other knowledge" is of two types. The first type is part-specific and machine-specific knowledge that is used by the industrial engineer to determine which DIPs are insertable by the machine. This knowledge is acquired from instruction, user manuals, and experience. By looking at the DIP part numbers for an assembly, the industrial engineer determines which DIPs are machine insertable based on DIP size and the number of DIP legs. The second type of knowledge specifies the sequence of DIP insertion locations on the printed circuit board. This information comes from a CAD system and is ported to the

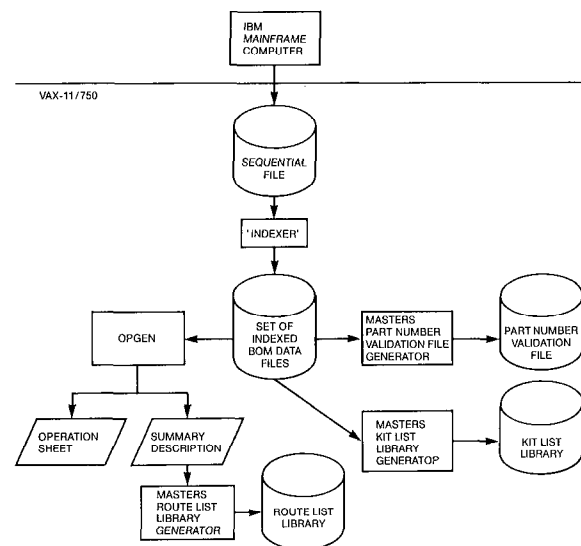


Figure 9. OPGEN Integration.

OPGEN host by magnetic tape. Once all this information is collected for a particular printed circuit board, it takes a typical industrial engineer four hours to produce the programming sheet and the chute loading sheet. OPGEN has been extended to generate the programming sheet and the chute loading sheet from similar input used by the industrial engineer.

### Current Integration Scheme

OPGEN exists on several computers. It is currently maintained (in Common Lisp) by the Hazeltine research laboratories on Theaetetus, a Symbolics 3600. Configuration control is managed both on the Symbolics and on a VAX/11-782. OPGEN executes on the VAX/11-750 that belongs to the Hazeltine manufacturing facility. The current OPGEN system integration with the manufacturing control system is shown in figure 9.

### Acknowledgments

We wish to acknowledge the following individuals who played significant roles in the development of OPGEN: Dave Martin, Bill Sylvester, and Ed Stork (experts); Paul Sakson and Jesse Karp (technical planning); Ray Masak, Randy Cope, and Sal Nuzzo (management); and Rod Rinkus, Andrew Shooman, and Jeff Rosenking (implementation). We also wish to thank Andrew Shooman for his help in reviewing the manuscript.

### References

- Bachant, J., and McDermott, J. 1984 R1 Revisited: Four Years in the Trenches *AI Magazine* 5(3): 21-32

- Barr, A., and Feigenbaum, E. A. 1983 *The Handbook of Artificial Intelligence, Volume 1*. Los Altos, Calif.: William Kaufmann, 193
- Brownston, L.; Farrell, R.; Kant, E.; and Martin, N. 1985. *Programming Expert Systems in OPS5*. Reading, Mass.: Addison-Wesley, 201
- Chandrasekaran, B. 1985. Generic Tasks in Knowledge-Based Reasoning: Expert Systems at the Right Level of Abstraction. Paper presented at IEEE Expert Systems in Government Symposium, 62-65
- Forgy, C. 1981. OPS5 User's Manual. Technical Report, CMU-CS-81-135, Dept. of Computer Science, Carnegie-Mellon Univ.
- Fox, M. 1983. ISIS: A Constraint-Directed Reasoning Approach to Job Shop Scheduling. Technical Report, The Robotics Institute, Carnegie-Mellon Univ.
- Frail, R., and Freedman, R. 1986. OPGEN Revisited: Some Methodological Observations on the Delivery of Expert Systems. Paper presented at IEEE Expert Systems in Government Symposium (forthcoming)
- Frail, R., and Freedman, R. 1984. Increasing Support Labor Productivity: Artificial Intelligence Applied to Process Planning. Conference on Simulation and Training Technology for Increased Military Systems Effectiveness, Society for Applied Learning Technology, 52-56.
- Freedman, R., and Sylvester, W. 1985. The Evolution of an Expert System for Process Planning. Paper presented at IEEE Expert Systems in Government Symposium, 328-334.
- Georgeff, M. P. 1982. Procedural Control in Production Systems. *Artificial Intelligence* 18(2): 179
- Hayes-Roth, F. 1985. Engineering Systems of Knowledge: The Great Adventure Ahead. Paper presented at IEEE Expert Systems in Government Symposium, 678-692.
- Locke, J. S. . Essay Concerning Human Understanding. Reprint in *Everyman's Library Book II*, ed. John Yolton, Chapter I. London: J. M. Dent & Sons.
- McDermott, J. 1980. R1: An Expert in the Computer Systems Domain. In *Proceedings of the First National Conference of the American Association for Artificial Intelligence*, 269-271
- Polit, S. 1985. R1 and Beyond: AI Technology Transfer at DEC. *AI Magazine* 5(4): 76-78.
- Rauch-Hindin, W. 1985. *Artificial Intelligence in Business, Science, and Industry, Volume II*. Englewood Cliffs, N.J.: Prentice-Hall.
- Ullman, J. 1980. *Principles of Database Systems*. Potomac, Md.: Computer Science.
- Van de Brug, A.; Bachant, J.; and McDermott, J. 1985. Doing R1 with Style. Second IEEE Conference on Artificial Intelligence Applications, 244-249
- VAX Lisp User's Guide, 1984. Technical Report AA-Y921A-TE, Digital Equipment Corporation

The AAAI solicits your assistance in nominating individuals to the positions of President-elect and Councilor. Elections for these officers will occur in May of 1987. Please submit your nominations to the following address:

Woodrow Bledsoe, Chair  
Nominations Committee  
AAAI  
445 Burgess Drive  
Menlo Park, California 94025-3496

## Call for Workshop Program

The AAAI has supported small workshops for the last several years. This support includes publicity, printing, office help, and subsidies for other expenses. \$5,000.00 is a typical subsidy, but up to \$10,000.00 may be considered. Any topic in AI science or technology is appropriate. Anyone may volunteer to organize a workshop on any topic. The organizer(s) should determine the topic, the date, the site, and the procedure for selecting papers and attendees. S/he should also decide whether preprints should be distributed.

Proposals for scientific workshops should be made to:

Professor John McCarthy  
Computer Science Department  
Stanford University  
Stanford, California 94305  
(415) 497-4430  
jmc@su-ai.arpa

For workshops on applied topics, applications should be made to:

Dr. Peter Hart  
Syntelligence  
P O. Box J620  
Sunnyvale, CA 94088  
(408) 745-6666  
hart@sri-ai

AAAI proposes that program committees give special consideration to papers that have been presented at workshops in choosing invited speakers for national conferences.

## AAAI-87 EXHIBIT PROGRAM

THE AAAI-87 EXHIBIT PROGRAM WILL BE HELD 14-16 JULY 1987 AT THE SEATTLE CENTER COLISEUM, SEATTLE, WASHINGTON

COMPANIES OR UNIVERSITIES INTERESTED IN EXHIBITING AT THIS YEAR'S NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE CAN OBTAIN INFORMATION AND APPLICATION FORMS FROM:

Mr. Steven Taglio  
AAAI  
445 Burgess Drive  
Menlo Park, CA 94025-3496