System generator for producing manufacturing applications

by Y. Hazony

The Application System Generator (ASG/pc) offers powerful capabilities for integrating the personal computer into problem-solving in manufacturing engineering. Implemented on Pentium™-class personal computers, it provides an easy-to-use, application-building platform for mathematically based solutions to a variety of engineering problems. The ASG/pc provides a master editor for incrementally developing the complete environment of an entire application. Such an application may entail hundreds of programs, most of which are automatically generated. These programs also include color-coded textual and graphic screen interactions, keyboard control, file management, and access to auxiliary equipment. The ASG/pc is designed for use by the application-domain engineer who will also be the end user of the technology. Facilities are provided for the integration of components developed through teamwork. Two applications of the ASG/pc are described: one is in the domain of rapid-response machining; the second is a system for rapid analysis and redesign of the factory floor. The ASG/pc was designed to facilitate an incremental, self-learning process.

This paper describes an engineering problem-solving platform, implemented for the DOS-based personal computer. The objective of the Application System Generator on the personal computer (ASG/pc) is to bring the process of conceptualization, design, and implementation of engineering systems within the grasp of the application-domain engineer. The aim is to reduce the task of designing and implementing a substantial application system to one person-year. These application systems are designed to deliver high productivity to the design-to-manufacture process. This goal is achieved by introducing a high level of automation to the iterative process of application system design and implementation.

The high productivity attained by the system described herein is a result of the convergence of advances in both hardware and software methodology. On the one hand, the emergence of 100-MHz-class personal computers (e.g., Pentium**, PowerPC*, and Alpha** processors) at an ever-decreasing cost, creates a new opportunity to develop highly interactive and computationally intensive problem-solving tools for science and engineering. On the other hand, progress in the methodology of graphics programming, augmented by a computer realization of the algebra of nested arrays,[1] provides an outstanding foundation and a high-productivity tool for the rule-based implementation of new application systems. The convergence of these technologies can serve an individual engineer in a way similar to that of the traditional slide rule, with many orders of magnitude of enhanced power. The focus on the individual, application-domain engineer requires that the new system be designed so that it could be mastered through an incremental process of self-learning.

The ASG/pc was designed to integrate the full seamless-design-to-manufacture (SDTM) process, including the placement of the conceptual design stage up

front.[2,3] Such a methodology inherently starts at an abstract conceptual stage and gradually evolves into a fully functioning SDTM system producing the actual physical end product. The evolution of this metamorphic process can be described in terms of a series of diagrams. This diagrammatic representation is based on an extension of the algebra of nested arrays. The diagrams represent an automatic implementation of the computer application system. The implementation diagrams evolve until the nested array, depicted by the diagrams, represents the fully operational system. The algebra of nested arrays used in this implementation is described briefly to explain its contribution to the high productivity attained in the development of new computer applications.

The ASG/pc provides an application-domain engineer with the capability to develop computer-based, customized, problem-solving tools. It provides an alternative to more traditional application-software development tools, which typically require the involvement of teams of application development specialists. These customized tools add up to a toolkit, with the possibility of merging it into an integrated, self-contained, engineering solution to a problem at hand. The initial set of tools consists of an ASG processor and a network of "editors." These tools address the programming requirements of the design and implementation of a rule-based engineering solution. Sample applications will be described, including "manpower" analysis.

Antecedents to the Application System Generator

The motivation for the implementation of the ASG on the personal computer was the power of the methodology demonstrated by the accomplishments of its predecessor system on the mainframe computer platform.[4,5] The latest application system reported, using the mainframe-workstation platform, was a system for the seamless-design-to-manufacture (SDTM) of marine propulsers.[3] This system required about one person-year to develop. It demonstrated a dramatic reduction in turnaround time between the design concept and a manufactured prototype. It reduced the time required for the full cycle of design-to-manufacture of a propulser blade from more than six months to under a week.

The need for a new system. The success of the SDTM system and the emergence of Pentium-class personal computers, combined with other interactive advantages of personal computers, points to the opportunity for higher productivity to be attained. This opportunity motivated the redesign of the methodology of the ASG/pc. However, the new approach must build on the initiative and ingenuity of the individual application-domain engineer. For an application system generator to fit the needs and working paradigm of such individuals, it has to provide powerful initial results in a relatively short time—on the order of hours and days. This requirement is necessary so that a case can be made for a more substantial investment of time, with much higher expectations for productivity enhancement. Consequently, a proposed problem-solving platform has to exhibit:

- A short initial learning curve in order to facilitate an "instant" preliminary success
- An incremental self-learning process, through which the motivated individual engineer could pursue a higher level of proficiency in the methodology
- A mechanism for "teamwork," providing for easy integration of modules developed individually, while maintaining recognition of the ownership of solution elements when a project develops into a "team effort"

Beyond a predecessor system. The ASG/pc entails a complete redesign and a major extension of a previously reported, mainframe-based system, the Expert System Generator (ESG).[4,5] The new system emphasizes integrating conceptual design into the process of system design. It includes new capabilities aimed at better handling of the generation and management of the underlying rule base. Furthermore, it takes advantage of the algebra of nested arrays to deliver higher efficiencies, both in the development and the execution of computer-based solutions to engineering problems.

Both the ESG and ASG/pc processors are implemented on the basis of data-driven-control-flow (DDCF) methodology.[4] The control flow of an application network was implemented through the concept of *key-controlled links*. Such links carry with them individualized sets of execution rules. In this implementation, links are controlled by *key-execution tables*, and an executed link is blocked if one of the relevant rules is violated.

The ASG/pc extends the conceptual scope of its predecessor to incorporate capabilities for

- Conceptual design
- Extended graphic programming
- Integrated run/edit/program-edit modes
- Automatic encapsulation
- Integrated archive for version management
- Topological mapping of rules on an application network
- Facility-focused rule development and management
- Streamlining of facility names and key designations

The new system embodies a collection of generic elements, e.g., screen configurations, textual screen interactions, graphic screen interactions, keyboard interactions, and file and input/output interfaces. It is built upon a collection of procedures and rules, which serve to automate most of the system design and application-building process. Text and graphic interactions can be combined on a single screen, and an extensive (and easy) use of color coding of both text and graphics is provided.

Mathematical foundation. A major advance of ASG/pc with respect to the original implementation of the Expert System Generator is in the use of a commercial implementation of the algebra of nested arrays.[6] This implementation provides a powerful set of mathematical tools for the structural and algebraic manipulation of such data. It serves as the mathematical base on which both the development system and the customized applications are built.

The term *nested array* implies having data structures, e.g., vectors, matrices, and scalers, which accommodate data items that may themselves have an internal structure. This internal structure may in turn include scalers, vectors, matrices, and nested arrays. Elements of an array may be of any type, e.g., textual, numeric, or Boolean, and the array would be of a mixed type.

The structure of nested arrays is suitable for networking the execution tables, the associated link functions, and the corresponding rule sets. The algebra of nested arrays consists of tools for the selective extraction and selective assignment of elements. These tools are integrated with a set of algebraic tools, borrowed from the domains of linear algebra, Boolean algebra, relational algebra, algebra of constraints, and set theory. The algebra of nested arrays integrates all these tools into one internally consistent set, with the addition of powerful generalizations.[6–9]

To understand the need for a complicated data structure, consider the task of managing one interactive screen, which is part of a network of screens designed to respond to the diverse requirements of a particular application. The particular structure used by the ASG/pc to describe a screen, which is made of N fields, consists of a nested $N \times 10$ matrix of data. The first column of the matrix includes data items that describe the location, dimensions, input/output (I/O) classification, and color attribute definition of each field. Each of these data items has the internal structure of a six-element numeric vector.

The second column of this nested matrix contains textual vectors that include the fixed text appearing in some fields on the screen, i.e., titles and instructions. These data differ from those associated with dynamic I/O fields, which require variable names to associate particular data with the corresponding fields. The actual $N \times 10$ matrix also includes logical (Boolean) data and the names of programs that contain rules regarding the input of legitimate data in the "open" fields on the screen.

The ASG/pc methodology was designed to accommodate the unstructured thinking process often associated with conceptual design. This process is best described by the concept of "irregular spider web" (ISW) networks, which allows the unrestricted transfer of control from almost any part to any other part of the application. The flexibility of the DDCF methodology [4] combines with the power of the nested arrays and the extensive use of rules to provide an ideal platform for the implementation of the ISW network structure. The ISW structure minimizes the need for human interaction and significantly reduces the opportunities for errors.

The original implementation of ESG[4] did not benefit at all from the advantages of nested arrays. A later version of the ESG[5] provided these advantages for implemented customized applications, without the ESG platform itself benefiting from it. This dichotomy was removed in the present implementation with the complete redesign of the methodology to take advantage of the power of the Pentium-class computer. The new system, the ASG/pc, takes full advantage of the algebra of nested arrays, both within the development system and the application domain.

The underlying hardware/software technology. In this paper, a general outline of the methodology employed by the ASG/pc is given, with minimum reference to the underlying software platform. The

premise is that the concepts described and performance attained could be duplicated, in principle, on any software and hardware platform.

Design of the ASG/pc

The Application System Generator is a rule-based system for the design and implementation of rule-based computer application systems. It consists of an ASG processor and a network of interactive editors. It provides the engineer with concurrent capabilities for the iterative top-down *conceptual* design and bottom-up *implementation* of an interactive, computer-based solution to a specified engineering problem.

The ASG/pc consists of an integrated set of three basic modes of operation.

A rule-based approach. The seemingly monumental task of building interactive application systems can be reduced to the task of a rule-based system, bringing about an orders-of-magnitude reduction in both time and costs.[4,5] The ASG/pc platform exemplifies such a system. It contains a set of interactive editors facilitating the implementation of a rule database that is part of the application control data array. Embedding the rule sets in the application control data structure enables a powerful simplification of the design, implementation, and maintenance of the rule database. Embedding the sets is done within the context of the underlying nested array platform, without requiring that the application developer be involved in its structural details.

The fact that the implementation of the nested arrays includes an algebra interpreter (the programming language APL2) simplifies the task of implementing the individual rules. The fact that rule grouping is inherent to the structure of the nested array eliminates some severe problems imposed by rule-based systems that maintain all rules in one pool. The tight mapping between rule grouping and the structure of the application eliminates the need to rely on other mechanisms to sort out the rules and maintain the consistency and integrity of the relevant rule subsets.

A master editor. The ASG/pc consists of an integrated set of three basic modes of operation: the edit and run modes and a program (function) editor. Consequently, in edit mode the ASG developer is no farther than two keystrokes away from the execution (run mode) of the new version of the application. Similarly, during run mode, the developer is always one keystroke away from returning to the edit mode. The program editor is directly accessible from within the edit mode of ASG, so that upon exit from the editor, the new version of the edited program becomes part of the current version of the system under development. This capability amounts to instant switching, back and forth, between application edit and run modes. It facilitates the gradual evolution of the structure of the application and the corresponding rule sets toward the final implementation.

The ASG/pc may be viewed as a master editor, designed to simultaneously handle a network of programs that constitute a complete application. A fully developed application may easily be comprised of hundreds of programs. In contrast, the program editor, which is invoked internally by the ASG and is provided by the underlying system, is designed to handle either one program or a "ring" of programs at a time.

Problem conceptualization. ASG/pc provides a problem-solving environment that accommodates the tentative nature of the initial conceptual stage of the problem-solving process. The incremental nature of the ASG development process, combined with the immediate toggling between edit and run modes, allows a gradual evolution from abstract definitions to an operational end product.

As an example, consider the case in which part of a mechanical assembly failed and would have to undergo an engineering change. The failing part was manufactured using a numerically controlled (NC) lathe, which is to be used for the production of the new part.

Table 1 represents a tentative activity list that can be developed as an initial approach to the design and implementation of such a system. It includes an initial problem-setup stage, which may include both parametric and graphic specifications or explorations of the problem, or both. The next set of activities

Table 1 A tentative activity table

Stage	Specifications	Development of Facilities	
Setup	PARAMETRIC	(Numeric interaction)	Parametric control
	GRAPHIC	(Graphic interaction)	Graphic control
Part	PROFILE	(Graphic interaction)	Profile startpoint
	POINTEDIT	(Graphic interaction)	Graphic profile edit
	NUMERIC	(Numeric interaction)	Numeric profile edit
Fixture			
Process			
Solution		(Textual interaction)	Solution summary

Table 2 Facility-table definition

Screen	Mode	Type	Short Name	Activity
SETUP	PARMTRIC	0T	SETUP	Problem setup parametric control
SETUP	GRAPHIC	0G	GRAPHIC	Problem setup graphic control
SETUP	SOLUTION	0T	SOLUTION	Solution summary
PART	PROFILE	0G	1stPOINT	Profile starting point
PART	PNTEDIT	2G	POINTEDIT	Edit points in profile
PARTTEXT	NUMERIC	0T	TEXT	Numeric edit of profile
FIXTURE	FIXTURE	0G	FIXTURE	
PROCESS	PROCESS	0T	PROCESS	

may include the development of facilities for the graphic initial definition of the new part profile, and for graphic and numeric profile-editing activities. Two more activities, dealing with fixture and process definitions, are included, waiting to be articulated later. The last activity addresses the final solution stage.

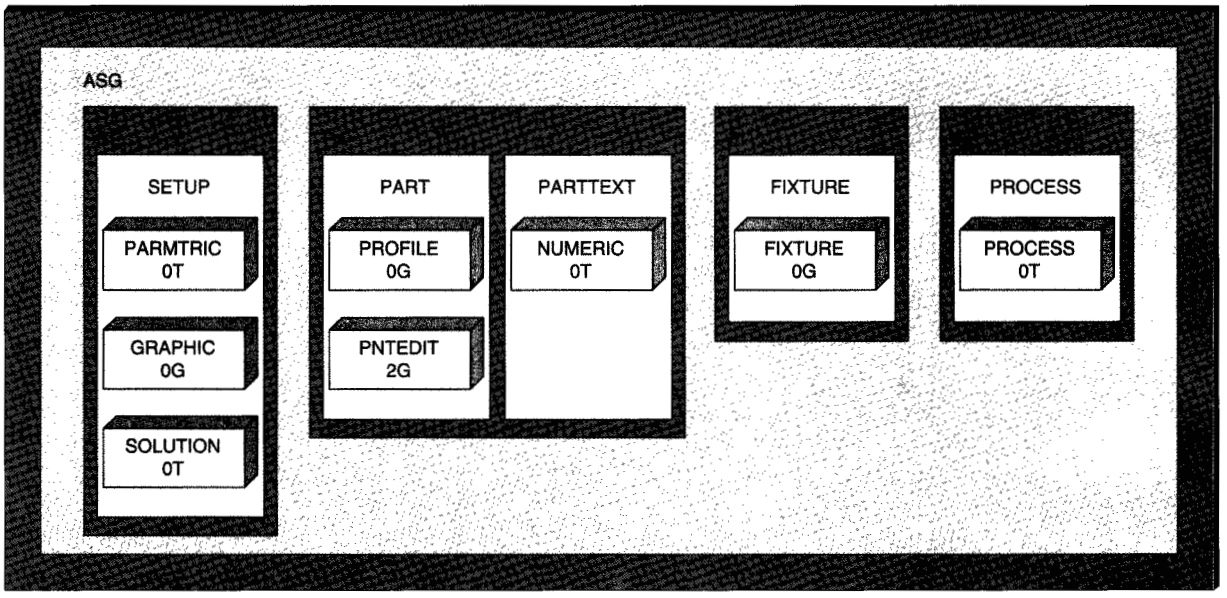
Problem articulation. The rule-based input convention of the facility-table editor converts the information included in Table 1 into a more detailed five-column table. The facility-table editor is used to enter data into Table 2, where each row of the table represents an ASG facility. It uses default terms for incomplete facility definitions, which were not included in the initial table definition.

The leading two columns in Table 2 include two distinct names that together constitute the name of an interactive facility. The name in the first column refers to the definition of the specific computer screen used for textual or graphic interactions, or both, required by the activity. The second name specifies the mode in which this screen is utilized, i.e., it represents the key-execution table. It specifies the key-response functions for the particular facility. The third column specifies the type of interaction, i.e., which of an available list of textual (or numeric, or both) or graphic input cursors is invoked by this facility. The fourth column includes a short "surrogate" name that is used by the system in referring to the specific facility.

The facility-table editor creates a data set that takes the form of a nested array and represents a structure for the application defined by Table 1. It includes some default data items, as well as "place holders" for missing data, which are to be specified during the evolving ASG development process. This nested array is used as drive data for the ASG processor mentioned earlier.[4,5] The data depicted in Figure 1 describe five interactive screens, which are shared by eight interactive facilities. The individual screens are tentatively assigned default screen layouts, and the respective facilities are provided with default definitions of key-execution tables.

The nested array illustrated in Figure 1 provides a simplified outline of the control data structure governing the application. It is generated in response to the entry of the *conceptual* list shown in Table 2. The data structure generated for this simple illustration constitutes a four-element nested vector. It is labeled as ASG, and the respective four elements are labeled SE, PA, FI, and PR, respectively. These elements are automatically generated as four "en-

Figure 1 Nested array representation of Table 2



capsulated" application editors. The term *encapsulated editors* will be discussed in more detail later.

Each of the four application editors is structured internally as a nested array. For example, the editor labeled SE consists of one screen, which is shared by three facilities. The editor labeled PA consists of two screens, PART and PARTTEXT, with the first screen shared by two facilities and the second one serving only one facility. The last two editors, FI and PR, serve as placeholders for data to be generated at a later stage of model articulation.

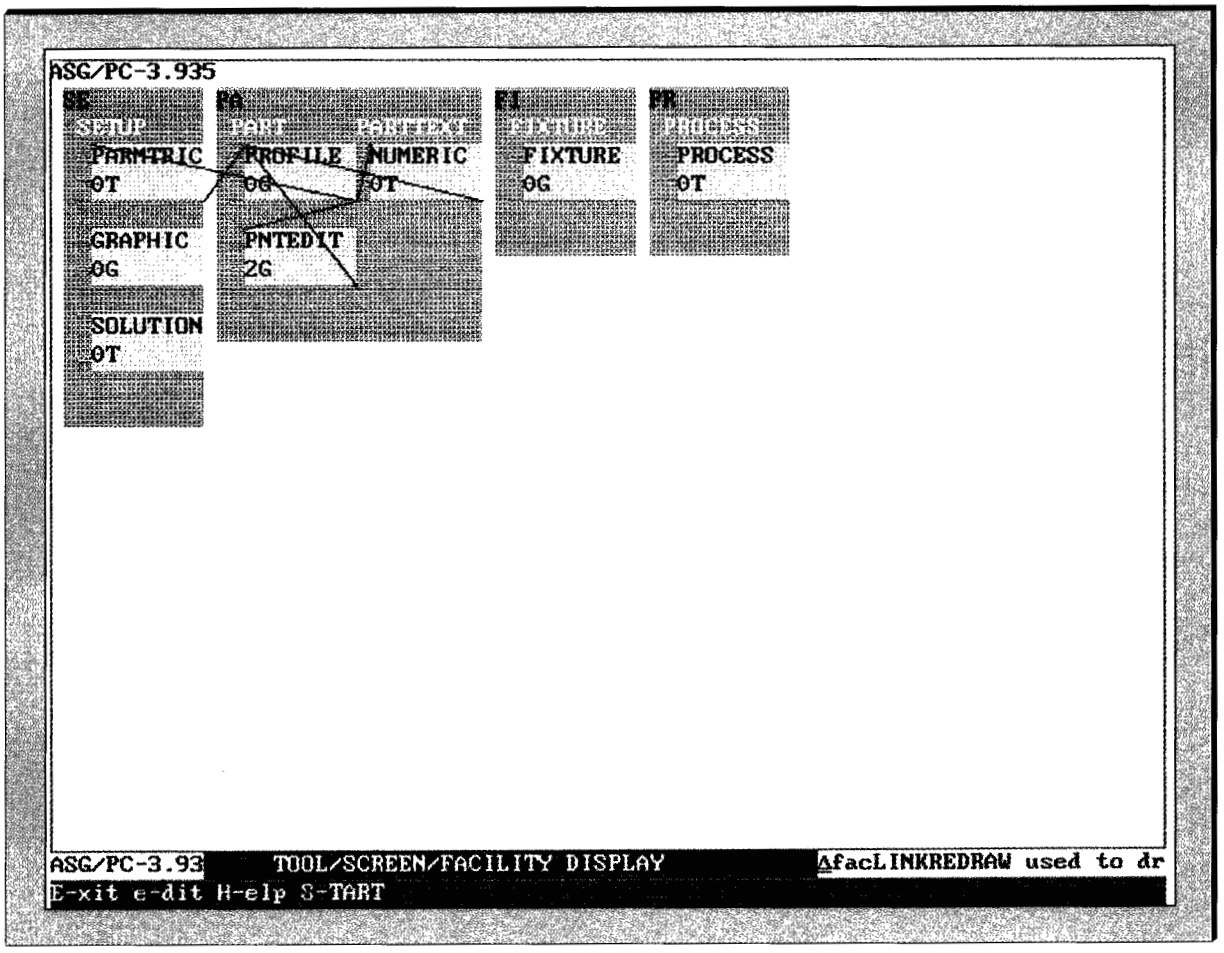
Each innermost data element shown in Figure 1 represents an ASG facility. It corresponds to a nested array of depth three, where the depth of a nested array is defined as the depth of the element that exhibits the deepest level of nesting. It defines the key-response functions specific to the facility. The layout and I/O interactions of each of the five screens, shown in Figure 1, are also controlled by a nested array of depth three, which is represented in the figure by the name of the screen, i.e., SETUP, PART, PARTTEXT, FIXTURE, and PROCESS.

The nested array depicted in Figure 1 is of depth six, but only four of the six levels of nesting are shown. The ASG employs a *dynamic definition* of the nested array, driven by the changes introduced to the facility table through the interactive application of the facility table editor.

An ASG/pc application may contain one or several editors, where each may incorporate one or several screens. Furthermore, each screen may be associated with one or several facilities. The facilities are associated with distinctly defined sets of interactions, each defined and controlled by a specific key-execution table, and by the type designation shown in the third column of Table 2.

The type designation of a facility is indicated by the second row in each of the innermost boxes shown in Figure 1. It shows that an editor may include a screen that is being exercised through various types of screen interactions. These interactions might be either textual or graphic, responding to the needs of parametric and graphic design modes. Various graphic interactions may be defined through a specific definition of the graphic cursor being used. Similarly, the type specification allows for the alternative attribute-designation of textual screens. These allow various I/O fields in a specific screen to open and close.

Figure 2 The application structure shown in Figure 1



Link specification. The concept of links between facilities is introduced to represent interactions in which control is transferred between facilities. It requires an extension of the original constructs of the algebra of nested arrays. The APL2 platform converts this high-power algebra into a "programming" tool via the "algebra interpreter" accompanied by a "program editor." The ASG/pc adds another layer to the interpreter in which the ASG processor recognizes appropriate data items that represent action through transfer of control between facilities. These data items were generated by the ASG during system design and implementation.

The data structure in Figure 1 may not be executed yet because it does not include any data items to affect transfer of control between facilities. These link data items include a key designation, a corresponding mouse button (optional), an originating facility, a target facility, and a link function.

Figure 2 is generated by an ASG editor to provide a color-coded view of the nested array that controls the particular application system under development. The data elements that represent the interfacility links are depicted in the figure as *graphic links*. A link emanates from the bottom right of the box representing the originating facility and ends at the top left of the one representing a target facility.

The individual links shown in Figure 2 are introduced through a graphic link editor provided by the ASG/pc. Links are sketched in graphically, and initial link

Table 3 A list of ASG editors

Facility-table editor
Key-execution-table editor
Link editors:
Graphic link definition
Textual link-table editor
Screen-layout editors:
Textual/graphic window partition
Graphic field layout editor
Permanent-text editor
Input/output editor:
I/O field-label editor
I/O attribute editor
Input-processing rule editor
Facility-focus editor:
Graphic processor definition
Encapsulation editor
Incoming link rules
Outgoing link rules
Function and rule editor
Archive manager
Help

functions are automatically generated. They hold link-specific sets of rules, and a link is automatically blocked if one of the rules is violated. The specific rule sets are gradually enhanced and extended during the evolutionary process of application development.

Automation of system implementation. The ASG/pc is using techniques of graphic programming to automate the implementation of the application system as defined by the process outlined previously. It creates all the data, programs, and rules required to have a running system. The ASG provides additional tools, referred to as editors, to proceed with the customization of the system to respond more specifically to the application requirements. These editors allow the developer to gradually modify the structural data, the generic programs, and the rules in order to follow the advances in model articulation. The list of editors and some functional details are provided in the next section. An additional set of tools associated with the requirements of graphic programming, encapsulation, and archiving are described in a later section.

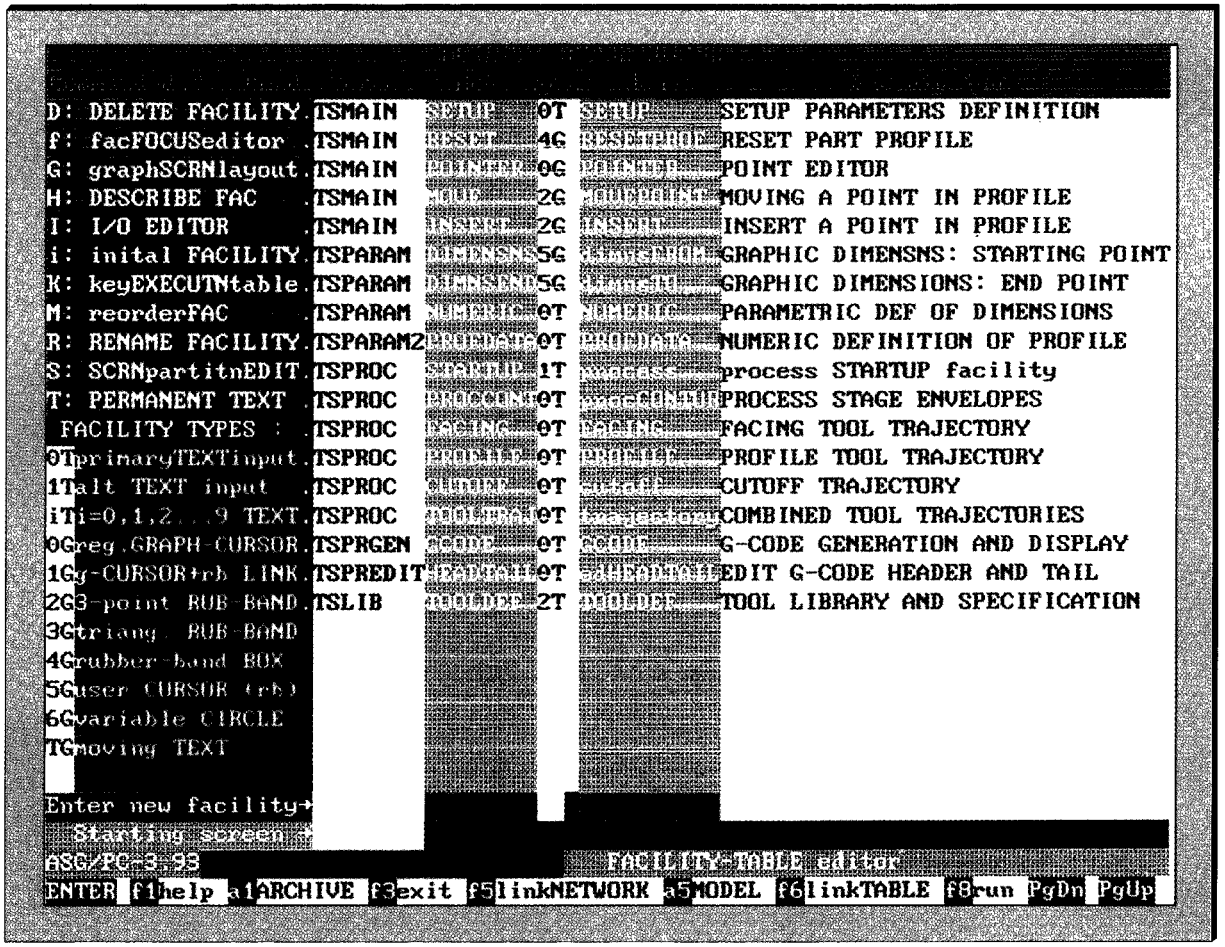
A sample application. The package described below was developed for demonstrating the principles of rule-based seamless-design-to-manufacture (SDTM)[2,3] in the classroom. It is used also as a tutorial for a manufacturing-systems design course. It permits a student to define the profile of a mechanical component of an engineering design that is to be machined on a turning center. It provides capabilities for sketching a design, "features" generation, and parametric specifications of a part. It also provides for the generative dimensioning of part-profile, as well as for stock and tool specification. It employs error-prevention rules and generative rules for automatic process design. The latter are used for the automatic creation of the manufacturing process, based on the capabilities of a specific machine and available tooling. The system identifies the feasible part of a design. It generates the appropriate tool trajectories, accompanied by the numerical control code (G code) needed to run the machining center and manufacture the desired part. The design-to-manufacture cycle time between conceptualization of the design and generated G code is as short as 5 to 15 minutes, depending on the complexity of the part. The evolution of this application of the ASG/pc is used to illustrate some of the main concepts and capabilities of the system.

ASG editors

In keeping with the ISW paradigm, ASG gives the problem solver a list of options with which to elaborate various data elements in no prescribed order. These options are provided through a network of ASG editors. Each of these editors is invoked through a single keystroke from within the facility-table editor and some of the other editors in the list. The accessibility of these editors from a given facility is governed by the specific key-execution table associated with each facility. A brief description of each editor is given in Table 3; however, the role of the individual editors becomes better understood through the evolving process of the ASG.

The facility-table editor. The facility-table editor constitutes the entry point to the ASG system. It is the means for defining a new facility as well as for changing a previously defined facility. It provides access to all other editors and provides a run key to start execution of the application at its present stage of development. Almost all other ASG facilities have a return key to this editor, thus providing a two-key sequence to start execution of the application under development. Similarly, all application facilities under development have a return key to the facility-table editor, thus offering a single-keystroke return to the ASG edit mode. These keys disappear when the ASG is started in run rather than edit mode. The facility-table editor shown in Figure 3 displays the five columns of the respective table.

Figure 3 The facility-table editor with table corresponding to SDTM sample application

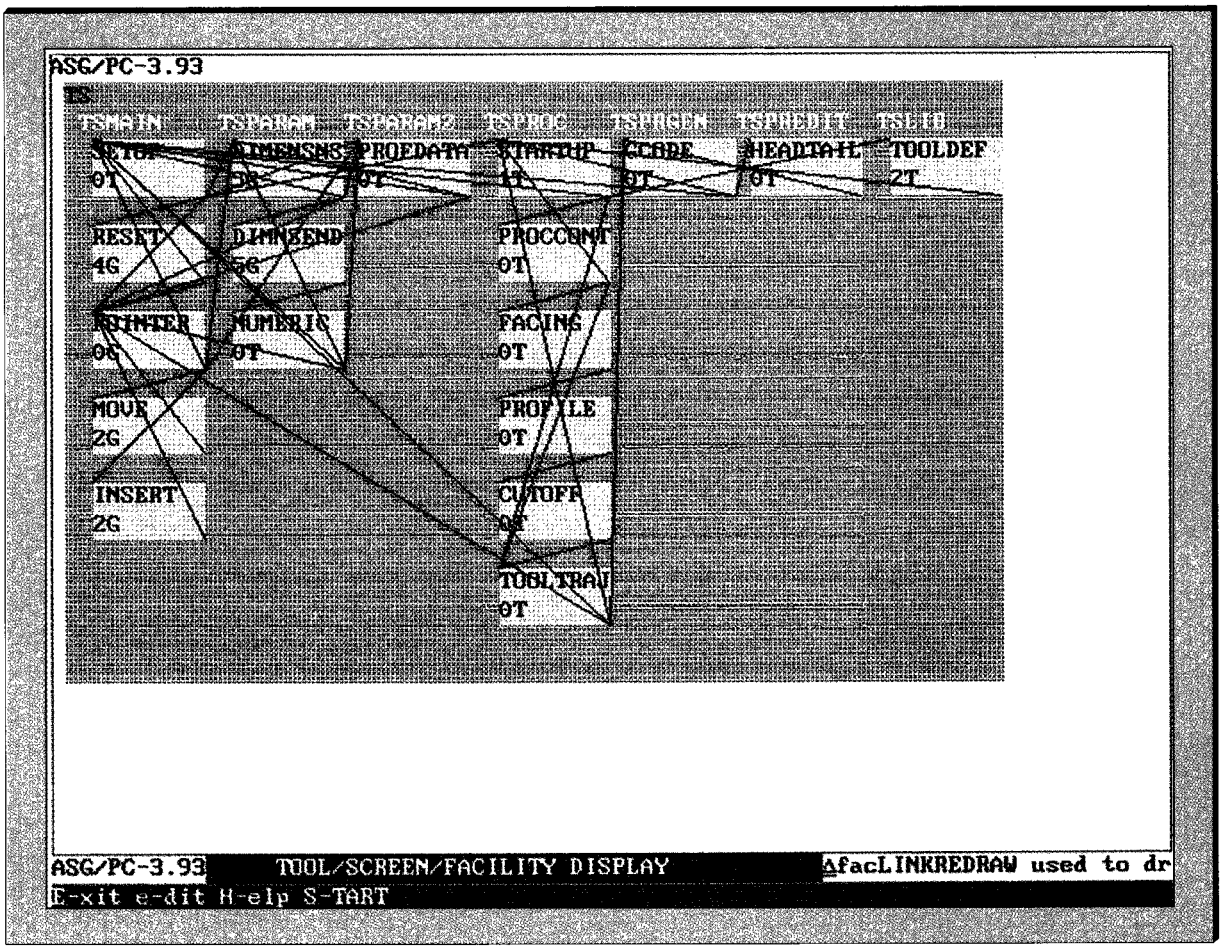


This editor is equipped with two sets of command keys. One set, shown at the bottom of the screen, provides command keys invoking local-response functions as well as links to other general editors. Another set provides access to editors applied to a selected facility. The list of commands available in this mode is depicted in the field (red) at the top-left part of the screen. These commands are exercised by placing the respective key in the command column to the left of the facility table, next to the appropriate facility, and pressing enter.

This dual set of command keys is a feature used by the ASG/pc but also available for the developing application. The actual table depicted on the screen represents the SDTM application outlined above. It has evolved from Table 2. The number of facilities increased to 18, representing significant progress in model articulation. The changes in the names used in the table are facilitated by the streamlining capabilities provided by this editor. These changes affect the overall structure of the system, which is automatically regenerated on the basis of the naming convention.

Figure 4 depicts the nested array model of the underlying data structure of the SDTM application. The system has evolved into four application editors. They are implemented as parts of one encapsulated package (labeled TS), as represented by the encompassing block (gray).

Figure 4 Nested array model of underlying data structure of SDTM application



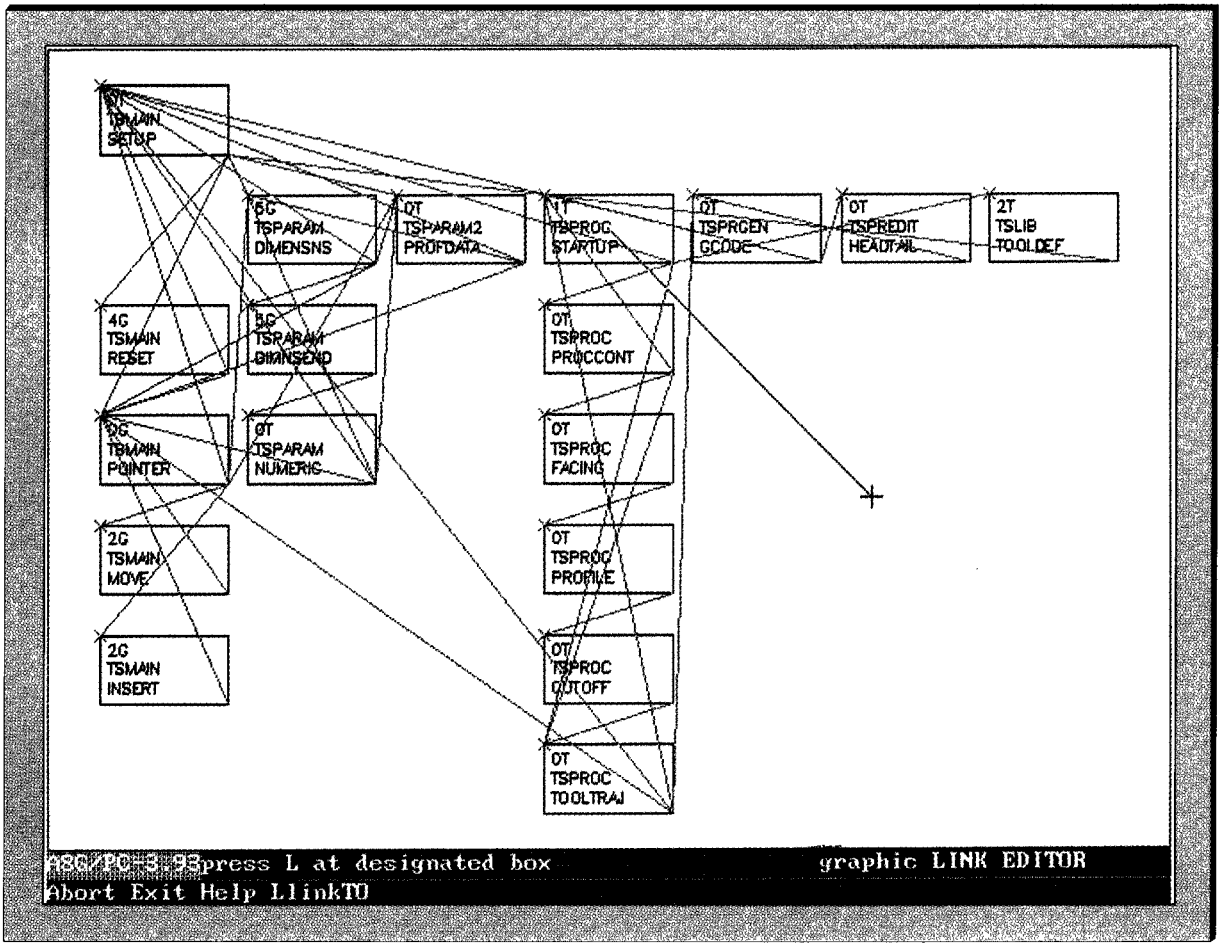
The four editors (each shown as a block) are:

- Main editor (TSMAIN), including setup and profile
- Dimensioning editor (TSPARAM)
- Process editor (TSPROC)
- Tool design editor and tool library (TSLIB)

The last editor consists of the tool-definition facility. This facility was initially developed as part of the process editor. However, it was removed from that editor to form a starting facility for a comprehensive tool-library editor. This change was accomplished using the streamlining capability, which is part of the facility-table editor. The restructuring of the underlying database is accomplished automatically following the appropriate change in the name of the facility.

The link editor. For an application to become "executable," links have to be specified between facilities as illustrated in Figures 2 and 4. Links are generated graphically via the graphic-link editor, which is accessible by a keystroke from several other editors. Each link is assigned a link function, which can be edited using a function editor through either the link-table editor or through the facility-focus editor. Links may be defined at any time once a minimum of two facilities has been defined. Once a set of links has been defined, the new system may be executed, at its present state, using the run key from the facility-table editor.

Figure 5 Graphic-link editor with two-point rubber-band cursor



The link editor consists of two subeditors: the graphic-link editor and the link-table editor.

The graphic-link editor is used to create new links or to delete old ones. This editor is associated with a graphic screen showing the full facility/link network of the application. The purpose of the following discussion is twofold. On the one hand, it illustrates the power provided to the system designer by ASG editors. On the other hand, it illustrates the kinds of design instruments that are made available for use by the application being developed.

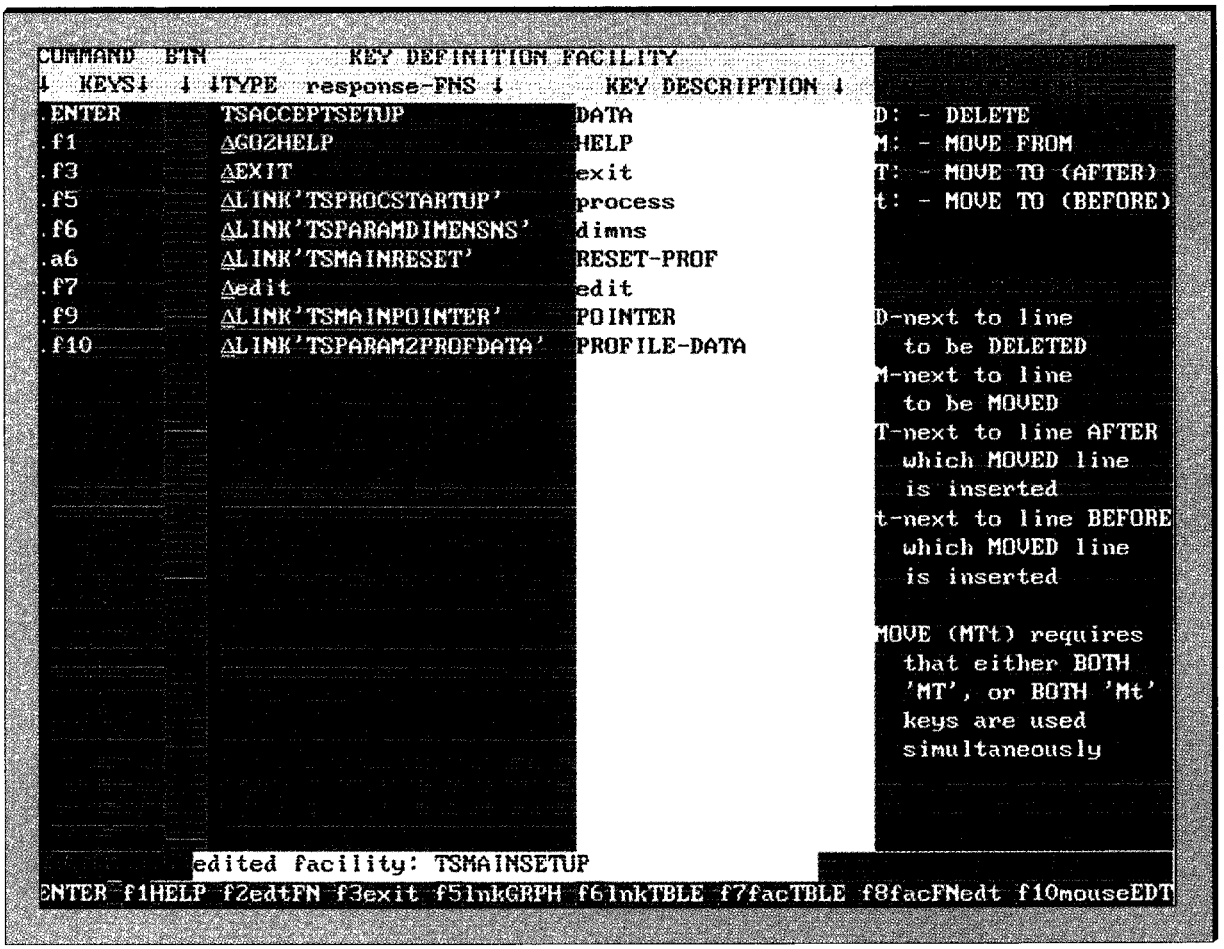
Figure 5 represents the second of a four-step sequence used to generate a new system link. The screen displayed by the first step is identical to Figure 5, with two exceptions:

1. The list of control keys in the bottom row is different.
2. The graphic cursor used by the respective facilities is different.

The four-step process for the creation of a new link is supported by four facilities. These steps consist of:

1. Graphic display of the facility/link network, accompanied by a simple moving cursor (type 0G), is used to point at the originating facility.
2. The fixed end of a two-point rubber-band cursor (type 1G) is automatically attached to the designated originating facility, while the moving end is used to point at the target facility.
3. The key-execution table is displayed on a textual

Figure 6 Key-execution-table editor with table corresponding to TSMAINSETUP facility



screen (type 0T), and an input field is opened on the screen for key designation. The table is displayed to avoid choosing a duplicate key.

4. The key-execution-table editor is automatically invoked (Figure 6—facility type 0T), displaying the new table, which includes the new link. This action permits editorial changes reflecting the consequences of the new addition to the table.

A default link function is automatically generated upon the creation of a new link. This function includes an initial set of rules governing the particular action. Any link is blocked if one of the specific rules is violated. It also includes a default definition of the graphic cursor, consistent with the type specified at the creation of the target facility.

Any particular link function is accessible with the link-table editor, once the application developer is ready to add, delete, or modify some of the rules it contains. The link-table editor is used for changing key designations and for renaming link functions so as to streamline such designations across an application. Additional access to the specific link function is provided by the facility-focus editor, to be discussed later.

The key-execution-table editor. The key-execution table (Figure 6) defines all the command keys available within a facility. They include program-function (PF) and command-column keys in the case of a textual-type facility, or regular keyboard keys in the case of a graphic-type facility. The entries in these tables

may represent either a local-response function or a link to another facility. A local-response function is accessible to the function editor by pointing at it with the alphanumeric cursor and pressing the function-edit key. The response function to an invoked link is a link function, which is accessible to the function editor either through the link-table editor or through the facility-focus editor. With this editor a mouse button can be selectively attached to an appropriate key. It also is used to change the key descriptors in the command line or field.

The screen displayed by this editor includes another example of the feature of a dual set of command keys serving a facility, which is available for use by the developed application. One set includes "global" keys (bottom of the screen), whereas the second set is utilizing a command column to the left of the table displayed. A separate output field lists the available commands (right, top of the screen). Another field provides instructions for the use of these commands.

The screen-layout editors. When a new screen is generated, it comes with a default layout of I/O fields. The screen-layout editors employ graphic techniques for easy and fast generation of such screen layouts. They are used to redesign this layout through the addition, deletion, and modification of fields. The group consists of three subeditors:

- The screen-partition editor
- The screen-layout editor
- The permanent-text editor

The screen-partition editor defines a graphic partition ("window") in a screen, automatically defining the rest of the screen as a textual partition. Such a mixed textual and graphic screen may be shared by both graphic and textual facilities. A screen may be associated with textual I/O accompanied by graphic output, or with graphic I/O accompanied by textual output. These classifications are done at facility specification time.

The screen-layout editor performs graphic generation, modification, and deletion of textual I/O fields within the textual partition of a screen.

The permanent-text editor is used to lay out titles and fixed instructions on output-only fields. The editor displays the actual screen, allowing the screen developer to enter the respective text wherever necessary. The text entered through this editor becomes part of the permanent structure of the particular screen. This permanent text could be changed, however, at any time by invoking this editor again. The screen shown in Figure 7 is used to illustrate the role played by these editors.

Figure 7 shows a graphic partition surrounded by three alphanumeric I/O areas that include:

- A title field on top (permanent text)
- A parametric input area to the left that includes five table titles (permanent text), five entry-description fields (permanent text), and five fields for parametric input/output
- A service area at the bottom for command-key description and messages

The process for designing the layout of such a screen has to be able to designate which of the alphanumeric fields is used for output only and which for input and output. Furthermore, the color coding of the various fields has to determined. These activities are performed by the input/output (I/O) editor described next.

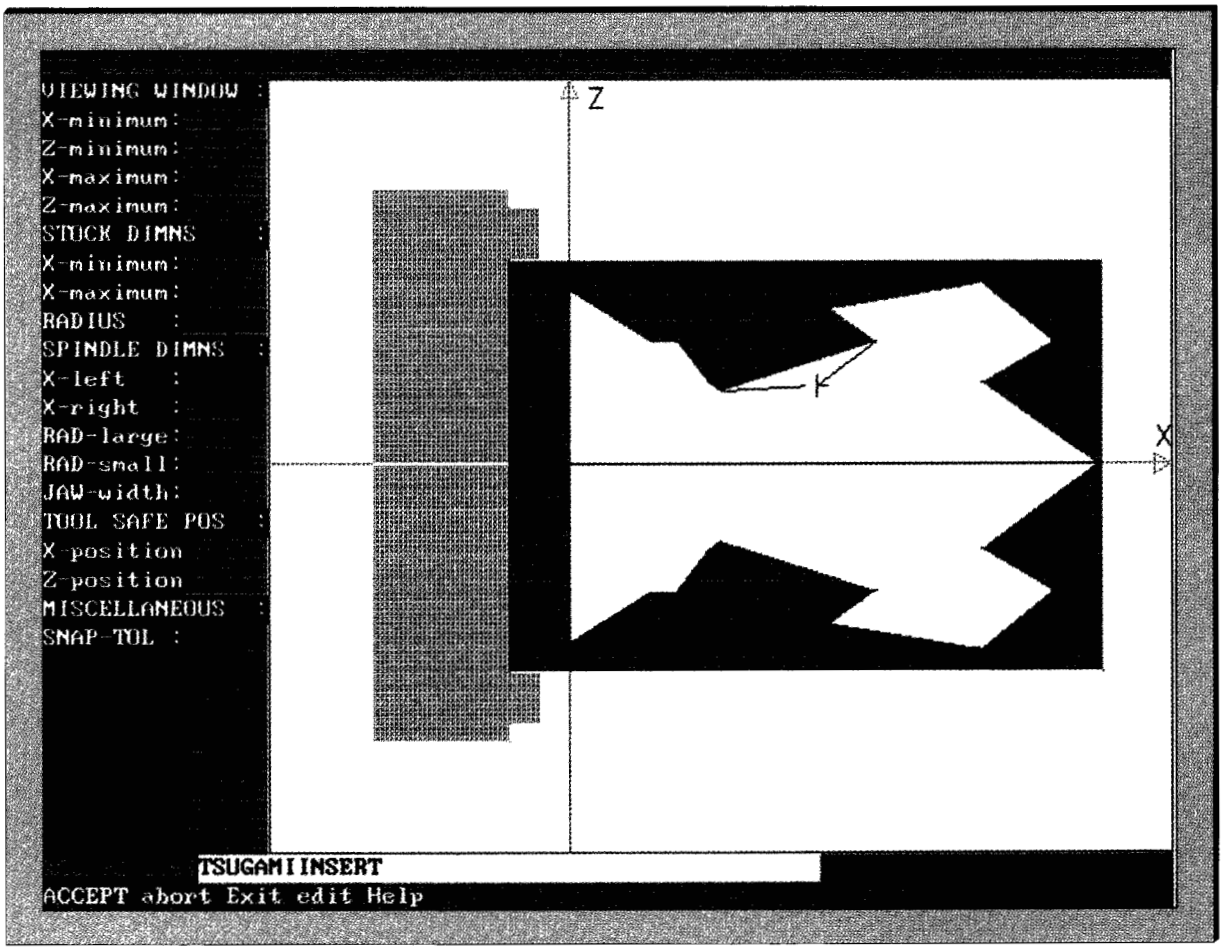
The I/O editor. The I/O editor is used to provide the different facilities with screen and data linkages, which create the context of each screen. These linkages determine which data are displayed in the various fields, and which variables would be updated by the data input through the particular open fields. The I/O editor consists of the following subeditors:

- The field-label editor
- The field-attribute editor
- The input-processing editor

These editors are accessible through one or two keystrokes from the facility-table editor. The ISW structure of ASG and the I/O editors make it possible to modify the interactive response of a currently working, previously defined screen at any time. Such modifications are necessary to meet changes in the system specification.

The field-label editor facilitates the designation of labels to the different I/O fields that make up a screen. The field labels are part of the nested array defining all I/O characteristics of a screen. These labels offer a contextual connection between the graphical layout of the fields on the screen and the data and functions associated with these fields. With the field-label editor, fields and labels can be visualized simultaneously so that a meaningful label may be assigned to each field. These labels are used in the field-

Figure 7 Screen layout for TSMAIN editor of SDTM application



attribute editor and input-processing editor, which do not provide a graphic display of the screen layout.

The field-attribute editor is used to associate various attributes with the different fields of a particular screen. These attributes include names of the variables associated with the output or input, or both, of each field, the type of data, color coding, and scrolling.

The input-processing editor allows the designation of groups of fields to be handled by specific input-processing (I/P) functions. These functions are structured to include sets of rules that will block acceptance of new data if a rule is violated. The input-processing functions differ from the previously discussed link functions in that they apply to any input action for all facilities sharing a particular screen. In contrast, local-response and link functions are facility- and key-specific.

The facility-focus editor. The facility-focus editor makes available access to all the functions and rules associated with a *particular facility*. It provides for the development of the rule-based application, without the need for interaction with the underlying structure of the rule database. The screen layout of this editor is partitioned into four color-coded areas, representing the following groupings:

- Incoming link functions
- Outgoing link functions
- Secondary editors:

–Local response functions
–Input-processing functions

- Auxiliary functions:

–Encapsulation entry rules
–Screen/facility specialized graphic displays
–Encapsulation exit function

The auxiliary functions do not require an initial specification. They become useful in the more advanced stage of application development. The encapsulation entry rules and the encapsulation exit function provide additional control over the editor-encapsulation process. The "specialized" graphic display functions become useful once the need for them is recognized.

The help editor. The ASG automatically generates help screens for each of the facilities created. The automatic help generator uses the description column in the respective key-execution table and the corresponding entry in the fifth column of the facility table to create an initial help panel for each of the facilities. This initial help panel supplies a skeletal description of all the command keys. The help editor provides an interactive edit mode of the help information. The ASG keeps track of the deletion and addition of control keys and updates the help information, respectively.

Additional ASG tools

Cursor interaction modes. A cursor type is specified for each facility using the facility-table editor at the facility-definition stage. The SDTM sample application takes advantage of several different type specifications: 0T, 1T, 2T, 0G, 2G, 4G, and 5G (Figure 3, third column).

ASG provides for ten textual, seven graphic, and one mixed-cursor (moving text) facility types. Consequently, when control is transferred between facilities by exercising a link, a prespecified cursor icon is invoked. Each different cursor is associated with the appropriate data structure and hooks to predefined I/O variables. This automatic software generation largely eliminates the need for some of the traditional programming tools during design and implementation of an application. Such programming instruments as branching or looping, or both, are essentially replaced by techniques of graphic programming.

The list of available facility types includes ten textual designations, iT (with $i = 0,1,2,\ldots,9$), that are used to switch between groups of active and inactive I/O fields in the same screen. This distinction is used to specify different groups of textual screen interactions without creating separate screens.

The list of graphic-facility types includes generic cursor icons that serve various functions:

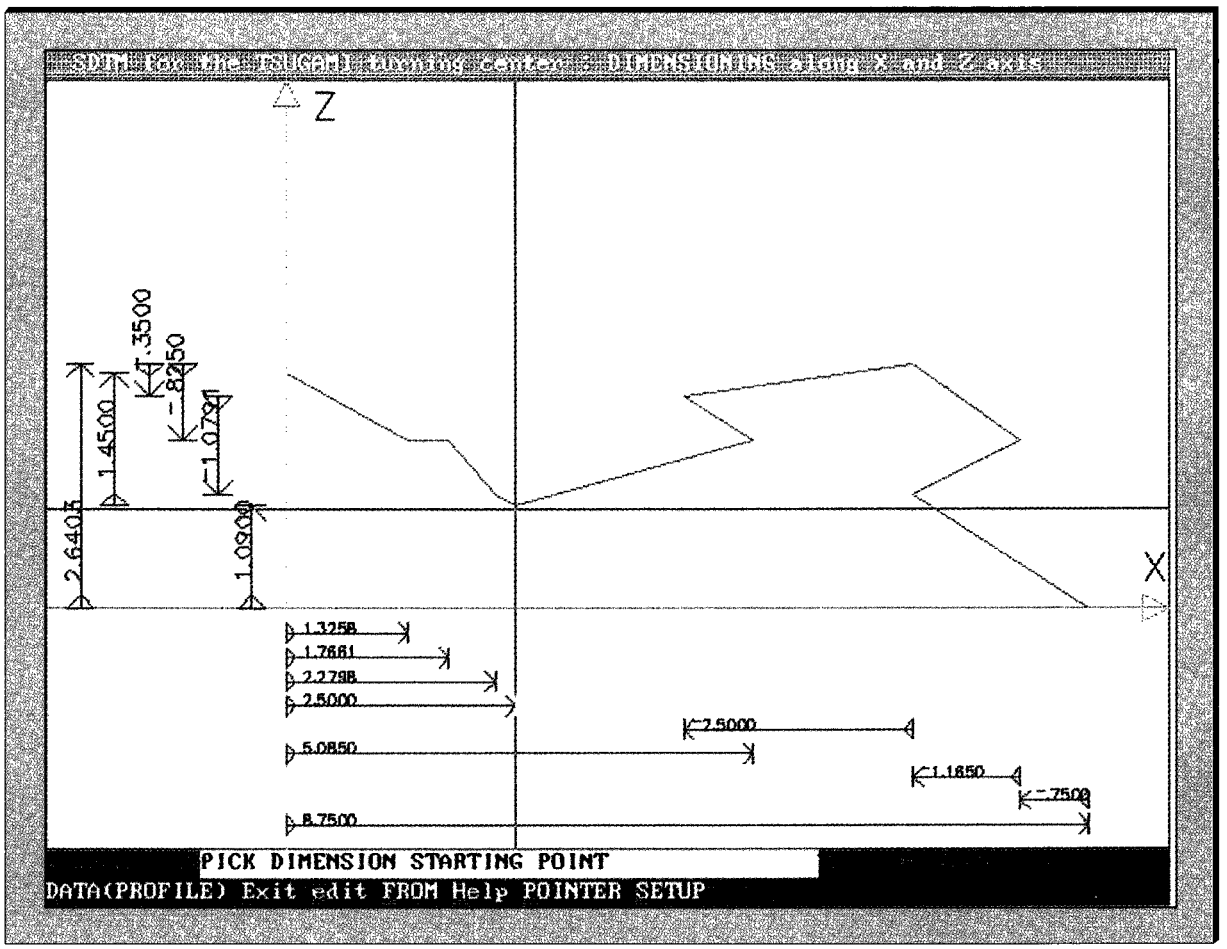
0G, 1G	Automates the process of graphical sketching in terms of points, lines, and polygons
2G,3G	Automates the sketch-editing process, providing for the moving or insertion, or both, of selected points in a sketch
4G	Creates a rectangular shape controlled by fixed and moving corners
5G	Provides user-defined icons. This type facilitates the arbitrary definition of moving icons, which may include discontiguous polygons as well as any number of "rubber-banded" links to fixed points. The size of the moving icon may be proportionally enlarged or reduced using the "alt(+)" and "alt(−)" keys. Similarly, the X−Y proportions may be varied using the "alternate" up, down, right, and left arrows
6G	Provides a circular icon controlled by the moving center position and the radius. The radius may be changed via the "alt(+)" and "alt(−)" keys.

The list also includes a mixed textual/graphic mode, TG, by which graphically generated text can be used as the cursor icon. This facilitates the moving and scaling of text for labeling and classification of data items.

The use of the 0G and 1G labels was described earlier in the discussion of the implementation of links (Figure 5). Examples of the use of the facility types 0T, 0G, 2G, and 4G are given in Figure 7 which is taken from the SDTM sample application previously outlined. It represents the TSMAIN screen shown in the first column in Figure 4. This screen is shared by the five facilities constituting the TSMAIN editor. The display corresponds to the TSMAININSERT facility. It is of type 2G and is last in the first column shown in Figure 4.

All facilities in this editor show a table of setup parameters and a graphic display. The first facility, TSMAINSETUP, is of type 0T. It provides for numeric input of setup parameters, accompanied by output-only graphic display. It displays an instant graphic

Figure 8 A user-defined cursor



presentation of the consequences of the parametric changes introduced through the numeric input fields.

In contrast, the other facilities are of types 0G, 2G, and 4G. They provide for output-only text accompanying an interactive graphic display. The graphic display depicts the shape of the fixture (spindle), stock, and part profile. (It is customary to show only a one-sided profile because of the spinning motion of the stock in the turning center.)

The TSMAINPOINT facility (type 0G) is used to pick a point or a line segment in the profile. It can also be used to delete a point. A link to a 2G-type facility activates the three-point rubber-band cursor to show a moving point attached to two fixed points via rubber-band lines. This interactive display shows the old profile and its modification simultaneously. The modification may be accepted by pressing an extra command key. Figure 7 shows the action of inserting an additional point to modify the profile (TSMAININSERT facility—type 2G). The new point is inserted into a designated line segment. The modification is affected by moving the cursor and pressing a command key for approval.

The 4G-type facility TSMAINRESET (Figure 4) is used to reset the profile to a rectangular shape. It provides a fixed-corner/moving-corner rectangular cursor to specify the new rectangular shape.

The 5G-type facility is used by the dimensioning editor (Figure 8), which is part of the SDTM sample application. This facility is using the combined vertical

and horizontal moving lines as a cursor. This cursor is used to line up a point or a line segment of the profile with the dimensioning data at the bottom and left side of the graphic screen. This particular cursor was implemented by the user-defined cursor capability (facility-type 5G).

The type designation of the target facility is used by the automatic link-function generator when a system link is implemented. The generator uses it to define and include a generic version of the particular cursor and the corresponding data. This is done to ensure that the particular link can be immediately activated on switching to run mode. Furthermore, the data provided with the generic cursor exemplify the particular data structure associated with each type of cursor. This likeness simplifies its replacement by data relevant to the actual application.

All graphic cursors are associated with data included in two global variables:

HERE—including the coordinates of the moving center of the cursor

POINTERDATA—including the data specifying the geometry of the particular cursor

These variables are specified in the incoming link function to preset the corresponding cursor. New data are included in these variables in response to cursor movement and which interactive key is pressed. The new data are available for processing within the local response function or the corresponding link function.

A graphic processor. The graphic processor is invoked by the ASG process cycle any time a key is pressed. A universal data-driven draw function is initially available and thus activated by the availability of the appropriate data. This universal draw processor may be preempted by a screen-dependent draw processor designed to respond to specialized requirements of a particular screen and all the facilities sharing this screen definition. The screen-dependent draw processor may be preempted by a facility-dependent graphic processor customized to the needs of a particular facility. Access to these graphic processors is available through the facility-focus editor.

Editor encapsulation. The term *encapsulation* pertains to a process that ensures that an application, or part of it, is implemented in a totally self-contained structure. This structure is codified in a way that distinguishes it from other encapsulated components, as well as the underlying ASG/pc editor network.

The automatic encapsulation is designed to facilitate the modular development of an application and to allow parallel efforts by a team of engineers. Similarly, it facilitates the separate archiving of an editor and the merging of several editors, developed for different applications or by different members of a team. Both the start-up facility and the facility-table editor provide a key to obtain a nested-array view of the application. This view separates encapsulated components (editors) that comprise the application.

Figure 9 shows the editor/screen/facility/link layout for the Factory Redesign Modeling System, which is discussed briefly later. The encapsulation structure is depicted by the three large rectangles labeled NE, WM, and DA. This structure is dictated by the input naming convention used by the facility-table editor.

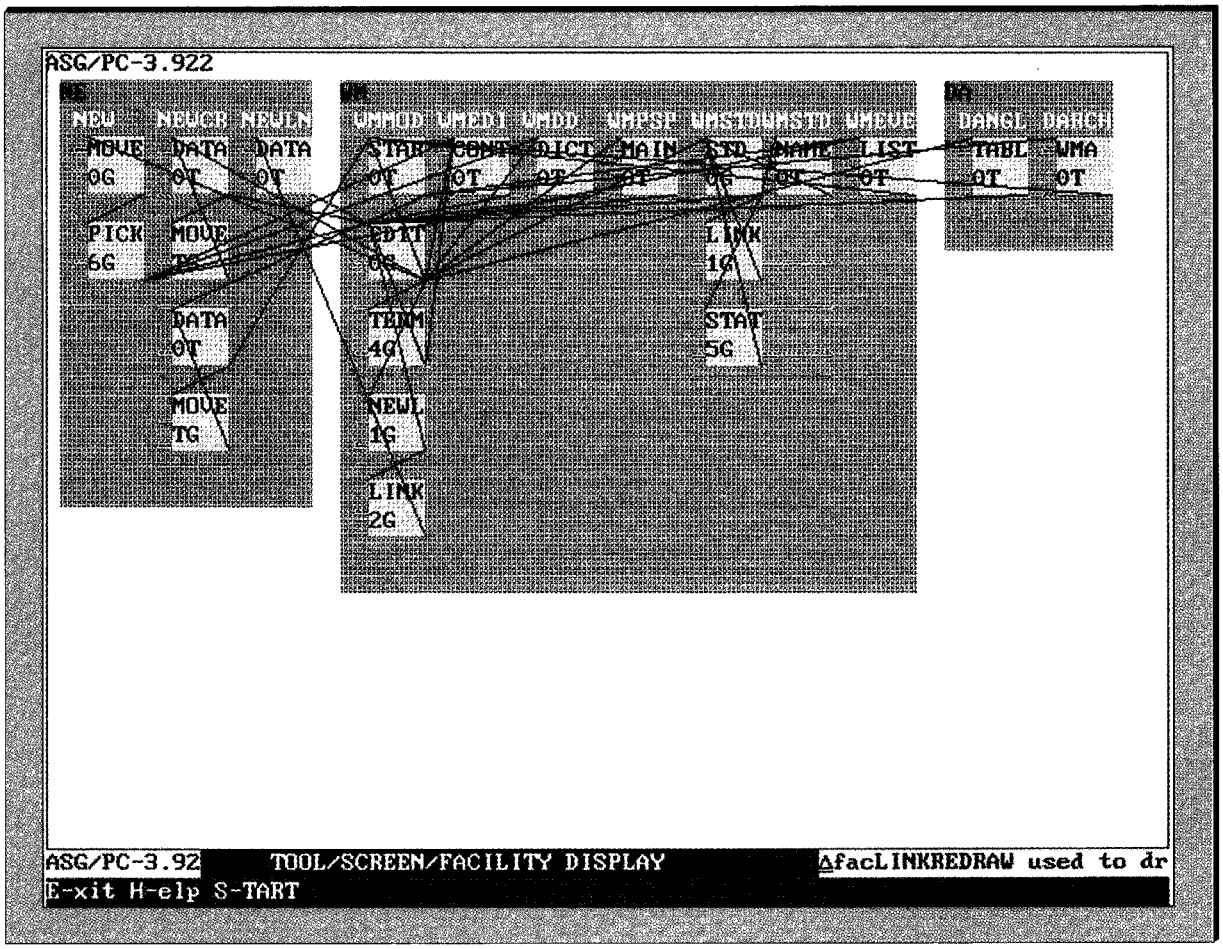
The archive, legacy-files, and activity-log facilities. The archive facility is built on the base of the DOS file system. It includes generic services for system and application backup. It creates and uses an ASG subdirectory, employing two distinct file extensions for system backup and application backup. It includes a command column for selection of the particular version of software to be restored (ASG system), loaded (application files), or deleted. The screen also includes a file-description field, which includes descriptors specified during invocation of the backup of either system or application.

The SDTM application (Figure 3) also provides teamwork services. To this end, it includes a facility for "user authorization" and for automatic management of "legacy files" and an "activity log." These activities can reside on the local hard disk drive, on a disk drive residing on a network server, or on the diskette drive.

Export of ASG/pc applications

The present version of the ASG/pc is running under the DOS operating system. It is used on 90-MHz Pentium computers with 32 megabytes (MB) of RAM. However, the system also runs comfortably on Intel 486 (33-MHz, 8 MB RAM) processors. At the low end, the ASG/pc runs on an Intel 386 (20-MHz, 4 MB RAM) processor. However, the development of an application such as described in the next section would be tedious using this configuration. The ASG software needs less than 1 MB of storage with application software not included. A "run-only" ver-

Figure 9 A nested array view of the editor/screen/facility/link diagram for Factory Redesign Modeling System



sion of the ASG occupies under 400 KB of memory and can be used to extend the usefulness of the lower-end computer configuration. The APL2/pc has the capability to package an application in an "execution-only" mode.[9] This capability permits shipping out legal "samples" for testing before committing the customer to purchase the underlying APL2/pc platform.

The commercial realization of the algebra of nested arrays, used in the work reported in this paper, is part of APL2/pc,[6] which runs under the DOS operating system. Powerful implementations of the algebra of nested arrays are also available for the UNIX**/C environment running on various "workstations," for Operating System/2* (OS/2*) (C-based), and for the IBM mainframe computer environment. All of these are available under various implementations of APL2.

The screen-control and graphic processors used by the ASG/pc were based on software provided by the commercial product (ap124 and ap207).[9] These processors facilitated the development of the easy-to-invoke, extensive color coding used for both textual and graphic interactions, as illustrated in the figures shown in this paper. The archive system described below is using the corresponding auxiliary processors (ap101, ap103, and ap210), providing access to the DOS file system. Dependence on the simultaneous use of these auxiliary processors makes the "porting" of the ASG/pc to other APL2 implementations a nontrivial undertaking.

Applications

Powerful applications of the predecessor system are described elsewhere.[2,3,5] In comparison, the ASG/pc was developed over the past two years, and thus the examples for its application are limited. Three such examples are discussed briefly in this section. The purpose of these descriptions is to illustrate the power of the ASG in the rapid creation of customized systems and to provide the reader with the "touch and feel" of the approach. A detailed description of these applications is beyond the scope of this paper. The examples include systems for

- The seamless design to manufacture on a turning center
- The Factory Redesign Modeling System, based on the Ward-Mellor modeling methodology
- The ASG, which was designed and implemented in itself

Seamless design to manufacture on a turning center. This package was customized for use in the classroom to demonstrate the feasibility of "instant-response" manufacture with numerically controlled conventional machines. The topic of SDTM has been touched upon earlier in this paper and discussed in more detail elsewhere.[2,3] The present application evolved out of this discussion, which was used as part of a tutorial in a course providing hands-on exposure to the topics of manufacturing system design. This tutorial was further developed to provide a hands-on demonstration of the power of rule-based, parametric, engineering design. It is used presently as a classroom assignment for a senior course on computer-controlled manufacturing.

Figure 10 illustrates one of the displays generated in this application. It shows the tool trajectories for a three-stage manufacturing process: facing, profile-cutting, and cutoff. It also shows the difference between designed and feasible (tool-dependent) profiles. The screen also includes the command-key specification row (bottom), the message field (second row from the bottom), and several title fields.

The effort invested in this package so far amounts to six person-weeks. It is an excellent hands-on demonstration of the underlying basic concepts, but it does not cover the full range of capabilities of the two-axis turning center. It is estimated that the effort to extend the package to cover a comprehensive set of tools and process rules will require six person-months, whereas the further extension of the package to include the capabilities of the third-axis of the three-axis turning center will require another six person-months.

The Factory Redesign Modeling System. A version of the Ward-Mellor modeling system[10] was implemented as a tool to achieve improvements in manufacturing productivity through factory redesign. The objective is to achieve improvement in productivity in the factory-redesign process itself. This achievement will make it a realistic exercise in a real-life operational manufacturing environment. The target cycle time for such a factory-redesign exercise is under a week.

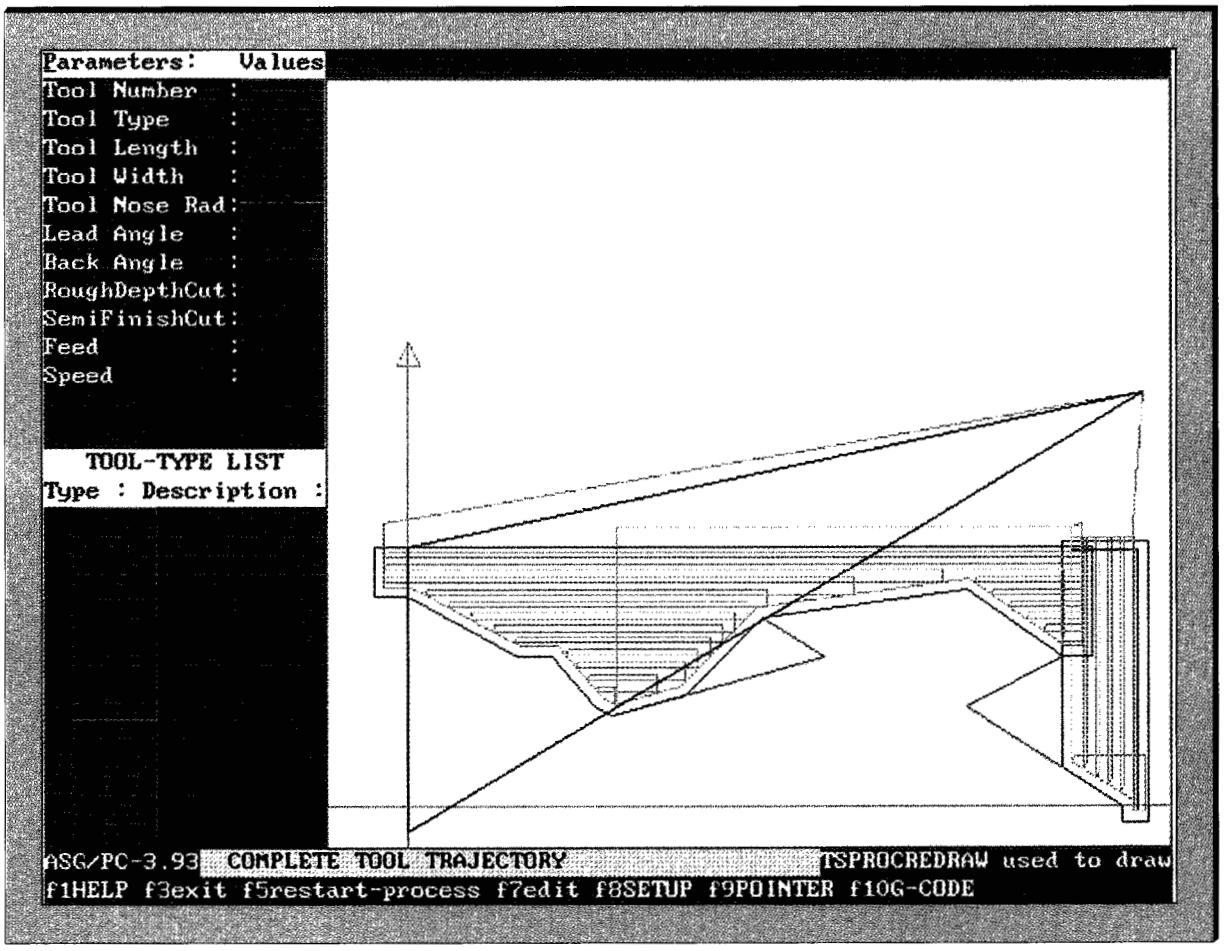
One of the graphical tools employed by the Ward-Mellor (WM) modeling method[11–14] consists of the data-flow diagram. It is a multilevel network of "bubbles" and "links." It utilizes bubbles to represent data transformation and links to represent flows. Figure 11 represents one of the displays used by the WM model to characterize the activity flow in a factory. A bubble is added to a "level" through the invocation of the circular pointer (facility type 6G), followed by scaling and placing it at a chosen location. At this point a textual screen with a graphic display window (facility type 0T) appears for the specification of the bubble label, color code, and attributes.

The specification of a new WM link entails a five-step successive process:

1. Invocation of a positional pointer (facility type 0G) to choose a link origin, which may or may not be a bubble
2. A "single rubber-band" pointer (1G) to pick a link destination
3. A "double-link rubber-band" pointer (2G) to determine the curvature of the link
4. A textual screen with a graphic window (0T) for the specification of a link label and link attributes
5. A TG-type screen having the link label as the moving cursor for placement at a nonambiguous location

Figure 11 corresponds to the third step above. It displays a level of the bubble/link network representing a WM model. It also displays the double-link, rubber-band cursor used to specify the curvature of the new WM link. Accepting the location of the curvature control point activates the facility used to specify the title and attributes of the new link. The newly accepted link is represented as an arc similar to all

Figure 10 Rule-driven tool-trajectory generation by SDTM system for two-axis turning center

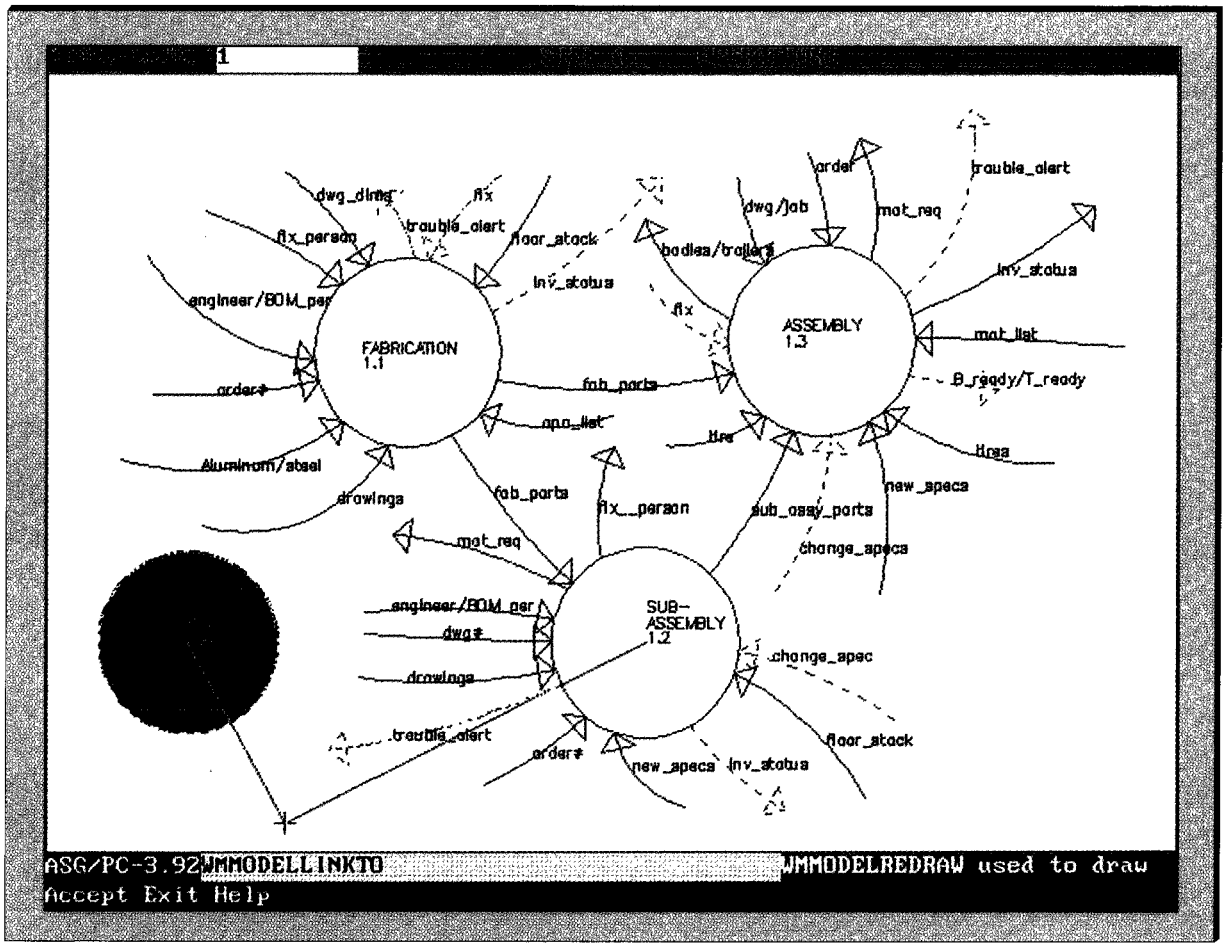


other links shown. The data shown in Figure 11 correspond to a particular factory.[12]

The WM system implemented so far includes rule checkers developed to enforce model completeness and data integrity. Rule checkers represent an extension to the original Ward-Mellor modeling system.[10] A mixed textual-graphic screen displays the corresponding network while a table shown on the left of the screen lists incomplete model-data items. This list was automatically generated by a rule checker. A command column at the left of the table permits the selection of any of the listed data items for the appropriate action. If the specific action complies with the rules, this data item is automatically removed from the list.

The effort of developing the present version of the WM-ASG modeling system is estimated at three person-months. This effort includes the stages of system conceptualization, system development, and model data entry. The system developed so far consists of 15 facilities sharing five screens encapsulated into three main editors (Figure 9). It includes 75 programs that handle all aspects of the computerized application. Most of these programs were automatically generated by the ASG/pc and subsequently upgraded by the developer through the use of the ASG editors. The effort required for the development of a comprehensive rule-based Ward-Mellor modeling system for factory redesign is estimated at one person-year.

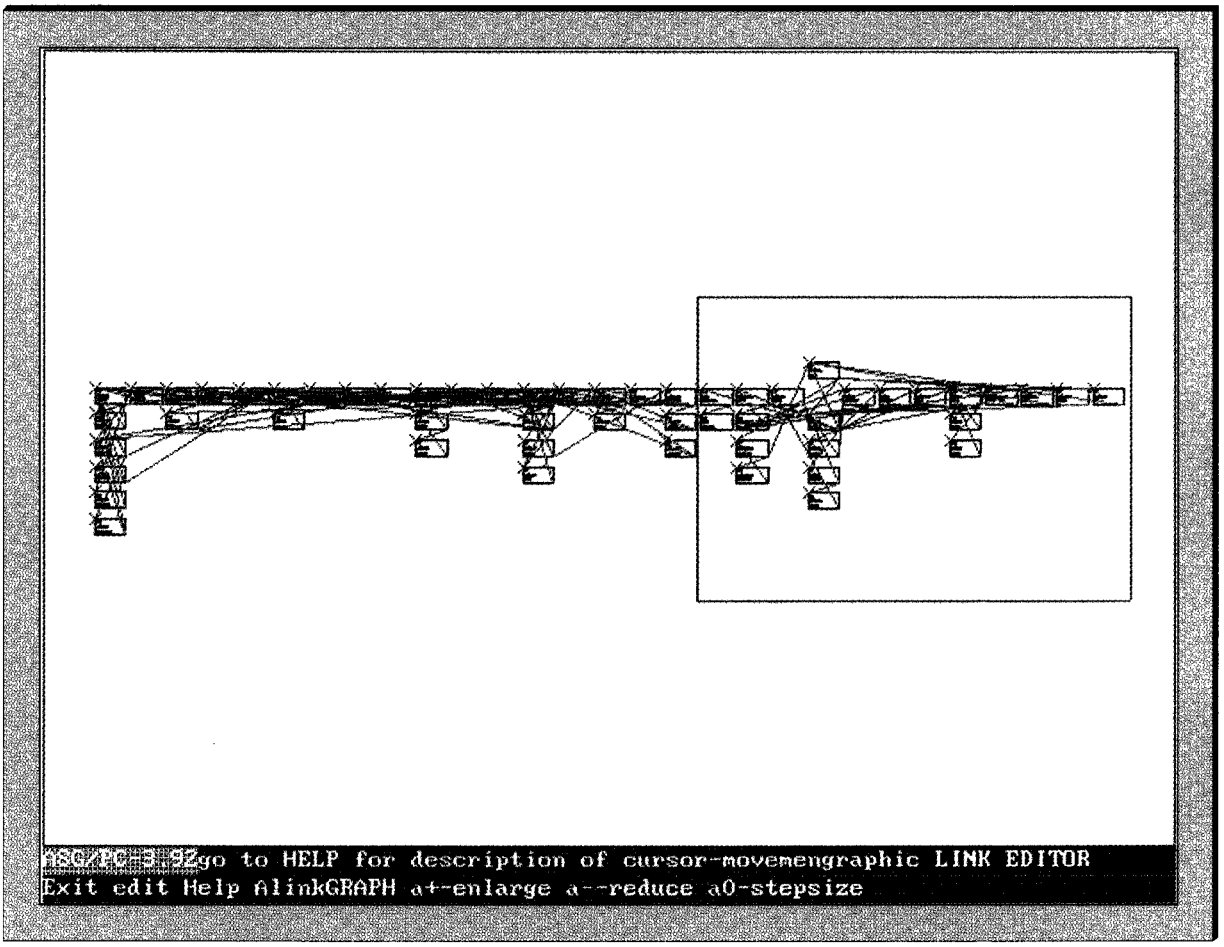
Figure 11 A data entry screen of the Factory Redesign Modeling System



The Application System Generator. The design and implementation of ASG/pc was carried out in itself using a "boot-strap" method. As such, it represents the most extensive application of the ASG/pc so far. An initial graphic processor was implemented and used to build a simple graphic editor for the graphic design of the I/O fields of an interactive computer screen. This primitive graphic screen-layout editor was used to develop the screen layout of an initial facility-table editor. This editor was designed for the definition of new facilities and for the management of a facility table. The two editors were used to design the link-table editor and the graphic-link editor. In this way the present version of ASG/pc evolved using itself as a system development platform. During this process, an ASG processor, the initial graphic processor, and the initial editors were gradually upgraded to accommodate the emerging requirements of the ASG/pc development process.

The present version consists of the 10 editors described earlier. It includes 16 screens shared by 30 facilities and connected via 87 links. Any application being developed may be viewed as an extension to the ASG-ISW network. New facilities can be developed and then added to the basic system. A system-control mechanism makes the system facilities and network invisible during the development of a new application. The same mechanism permits the reassignment of newly developed facilities to become part of the underlying system itself. Figure 12 illustrates this evolution process. It shows the network,

Figure 12 Complete facility/link network



including both the facilities that constitute the development platform itself and those representing the evolving new extensions and applications. The subnetwork of the newly developed facilities are shown in the rectangular zoom box. The system-control mechanism determines which part of this combined network will be encapsulated as part of ASG/pc and thus become invisible to the application developer or user. The boundary may be perceived as an arbitrary vertical partition line that may be moved to accommodate new capabilities.

The underlying system variables and functions are not locked (hidden), but they are prefixed by identification characters. In this way the system is protected from inadvertent changes. However, in the event that a system function or variable is compromised, a system restoration mechanism is included as part of the archive facility.

The total development effort invested in the ASG/pc is estimated at two person-years, not counting experience with its predecessor mainframe system. This amount of work represents an extremely low figure for a system having such power and capabilities.

Discussion

A major issue of concern with the methodology described in this paper is the very high level of complexity of the hardware and software platform utilized. It is clear that for this technology to gain

credibility in manufacturing engineering, the details of the hardware configuration and of the underlying operating system must be hidden from the user. At its present state of development, the ASG goes a long way toward accomplishing this goal. To this end it requires that the application developer acquire a certain level of proficiency in the algebra of nested arrays. However, the availability of commercial computer implementations of this algebra, in the context of an application development platform, makes it susceptible to an incremental self-learning process.

A related issue is the use of APL2/pc as an underlying programming language that matches the power and sophistication of the mathematical methodology. Although interfaces exist for calling programs written in C, FORTRAN, or Pascal, attempts to do so will very quickly compel the user to invest in the APL2 notation. The combination of nested arrays and graphical programming, offered by the ASG, makes APL2 programming *per se* to be a rather small component of the system development process.

Maintenance of such sophisticated computer systems and networks is still a bottleneck. A qualified, adequate organization is necessary to maintain network reliability compatible with the requirements of manufacturing. This problem is critical, independent of whether or not ASG/pc applications are being used on such a network.

Another issue touched upon is that of accommodating and supporting teamwork. Although the ASG was designed to facilitate the work by an individual engineer, the encapsulation mechanism described previously is designed to support the integration of individually developed modules into a comprehensive application. It is accomplished while maintaining the identity of the builders of the original components. The encapsulation mechanism used in ASG/pc may be viewed as "coarse-grained object-oriented programming," where this phrase is used in a very loose sense. It provides some of the advantages offered by object-oriented programming without the constraints imposed by such programming environments. This difference is significant particularly for applications that could benefit from a highly mathematical approach.

The teamwork support extends beyond encapsulation to include facilities for user authorization, activity log, legacy files, and an archive. The different data logging activities serve different purposes, and these facilities provide the foundation for extensive support for a team environment. However, large-scale teamwork activity could benefit from the advantages of an interface to an adequate commercial database management system.

Inspection of the various ASG/pc screens displayed in this paper reveals a different "touch and feel" than the window-style screens prevailing in contemporary personal-computer applications. The main differences are,

- Only one "window" is open at any time.
- Each "window" occupies the full screen.
- Switching between windows is through single key control.
- Switching between windows is facilitated by an irregular spider-web network rather than a tree network structure.
- There is extensive (and easy) use of color coding.
- There is extensive use of *various* graphic cursors as design and programming tools. These cursors are provided by the ASG/pc system or easily generated by the application developer.

Experience with student developers reveals that, given the facility, the design of a facility display evolves into one with an extensive amount of coherent data on a screen. This design is often accompanied by a wide choice of options through numerous control-key designations. It all has the effect of minimizing the number of interactions required for the completion of a task.

An alternative "touch and feel" could be achieved if the screen-control mechanisms provided by the commercial product APL2/pc were augmented to include interfaces to Windows**-style screens. An example of such a possible enhancement could be an interface to a popular product like Visual Basic**. Such an interface could serve as a substitute to the underlying mechanisms provided by the ap124 and ap207 auxiliary processors. However, Visual Basic and the standard auxiliary processors by themselves provide acceptable tools for the development of very simple applications that may be served by several "facilities" or "windows." Both methods become tedious and unyielding when a much higher level of complexity is called for, especially where a rule-based approach could be advantageous.

System-development tools such as Visual Basic may be viewed as a potential substitute for the auxiliary processors ap124 and ap207 used by the ASG/pc system. However, such a substitution will lead to an al-

ternative but entirely different interactive paradigm. In contrast to the touch and feel described above, a Visual-Basic-style system generates numerous overlapping windows. For a complex application, the overlapping may be viewed as an extensive clutter of information on a single screen. This clutter often requires a substantial involvement by the user, who is called upon to clean up the screen by selectively closing unnecessary windows.

Ideally, both types of screen interfaces should be made available, giving the user or developer a choice between two alternative styles. It would be even better if both interfaces are made to coexist under ASG/pc, to be used within the same application. The choice between the alternative styles may be a consequence of performance considerations or individual stylistic preference, or both. In both cases, however, the ASG/pc would provide a higher-level productivity tool. The Visual Basic option was not available during the development of the present version of ASG/pc and is not accommodated by more recent APL2 products.

The rapid development of computer technology makes it extremely difficult to provide a comparative performance evaluation with respect to past systems. The Pentium-type personal computer is substantially more powerful than the mainframe computer used for the development of the predecessor system of the ASG/pc. Furthermore, the personal computer is used as a dedicated machine, whereas the mainframe computer was employed as a time-sharing system. In addition, the projects addressed within the present implementation are different from those implemented in the past. Since the computer platform, the ASG/pc environment, and the application projects have all changed, it is impossible to provide a quantitative comparison between past and present performance.

The design objectives for the ASG/pc were substantially more ambitious than those for its mainframe predecessor, the ESG.[5] Furthermore, experience gained from past implementations was taken advantage of, and thus the ASG/pc may be viewed as the next generation of the ESG. A brief summary of the differences between the two implementations was given in an earlier section of this paper.

Performance evaluation may be provided, however, in terms of the definition and attainment of new performance objectives. The Factory Redesign Modeling System, briefly described earlier in the paper, was undertaken to verify a two-level objective:

- Application system implementation has to be feasible as part of a master's degree thesis project, imposing a time constraint of one year, which must allow for both student training and system implementation.
- The resulting modeling system should make it possible to implement a small-scale factory model, of the kind described by Ebner, Sharara, and Torres,[11,12] within 30 hours.

Both targets have been achieved as reported by Ebner and Godika.[13,14] Similarly, the SDTM system described earlier is used as a classroom exercise in an ASG/pc-based system design course. Furthermore, the ASG/pc was implemented "in itself" single-handedly, over a period of two years.

The system described here provides rapid-response solutions to emergency situations. The application systems implemented in the past all fell into that category.[5] For a system to be effective in this domain it has to display both time efficiency and effectiveness in adapting to the diversity of requirements posed by manufacturing applications. In addition, the ASG/pc application development platform is aimed at the end user being both developer and user of an application. This dictates a very short learning curve and reliance on further improvements through incremental self-learning. It is important to give an engineer the opportunity for a short-term initial success story, which can be used to justify further investments in time and equipment for the implementation of the next step.

Concluding remarks

The Application System Generator, implemented on Pentium-class personal computers, offers powerful capabilities for the integration of the personal computer into the problem-solving process for manufacturing engineering. It is an easy-to-use, application-building platform for mathematically based solutions to a variety of engineering problems. The ASG/pc is designed for use by the application-domain engineer, who will also be the end user of the technology. This goal is attained by software automation reducing the need for teams of application development specialists. The focus on the manufacturing engineer, both as an application builder and an end user of the technology, dictates a design methodology that facilitates an incremental self-learning process. This is signif-

icant in view of the fact that higher productivity is attained by increased use of tools of applied mathematics, such as the algebra of nested arrays.

The continued advancement in hardware technology is manifested in the emergence of new and faster processors and in multiprocessor configurations. In addition, advertised new versions of the commonly available operating systems would provide access to multitasking, multiprocessing, and high-speed networking. This trend is matched by a similar evolution of large-scale parallel machines in the direction of coarse-grained parallelism using a network of small machines. These trends introduce "coarse-grained parallelism" as an option for manufacturing applications.

These trends emphasize the need for application-development platforms that can harness the ever-increasing power of computers. An exploratory implementation of a "server network generator" has been described elsewhere[5] as an extension of the predecessor to the ASG. The ASG can be extended to encompass the functionality of the server network generator and thus demonstrate that such a technology can be realized and further developed.

Acknowledgment

The author expresses his appreciation to Merrill L. Ebner for numerous discussions and a critical reading of the manuscript.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Intel Corp., Digital Equipment Corporation, X-Open Company, Ltd., or Microsoft Corporation.

Cited references

1. T. More, *A Theory of Arrays with Application to Databases,* IBM Cambridge Scientific Center Report G320-2106 (September 1975). Also, *Notes on the Diagrams, Logic and Operation of Array Theory*, G320-2137, IBM Corporation (September 1981).
2. L. E. Zeidner and Y. Hazony, "Seamless Design to Manufacture (SDTM)," *Journal of Manufacturing Systems* **11**, No. 4, 269–284 (1992).
3. Y. Hazony and L. E. Zeidner, "Seamless-Design-to-Manufacture of Marine Propulsers: A Case Study of Rapid Response Machining," *Journal of Manufacturing Systems* **13**, No. 5, 333–345 (1994).
4. L. Zeidner, Y. Hazony, and A. C. Williams, "An Expert System Generator," *IASTED Journal of Control and Computers* **15**, No. 1, 22–33 (1987).
5. Y. Hazony and L. E. Zeidner, "Customized Systems for Engineering Applications," *IBM Systems Journal* **31**, No. 1, 94–113 (1992).
6. M. L. Tavera, M. Alfonseca, and J. Rojas, "An APL System for the IBM Personal Computer," *IBM Systems Journal* **24**, No. 1, 65 (1985).
7. *APL2/pc, Version 1.02* (Part No. 62429636), IBM Corporation.
8. A. D. Falkoff, "The IBM Family of APL Systems," *IBM Systems Journal* **30**, No. 4, 416 (1991).
9. J. A. Brown, S. Pakin, and P. R. Polyvka, *APL2 at a Glance*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1988).
10. P. T. Ward and S. J. Mellor, *Structured Development for Real Time Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1985).
11. M. Sharara and F. M. Torres, *Factory Redesign at Hackney and Sons*, Graduate Project Report, Boston University, Boston, MA (May 1994).
12. M. L. Ebner and F. M. Torres, "Factory Redesign Using Structured Modeling: An Example," *9th Conference with Industry*, Center for Manufacturing Systems Engineering, Lehigh University, Bethlehem, PA (May 1995).
13. S. Godika, *A Compact Small Model Implementation of SART Process Design*, M.S. Thesis, Boston University, Boston, MA (September 1995).
14. M. L. Ebner, "Use of SART Models in Manufacturing Process Design," for submission to the *IEEE Proceedings in Engineering Management* (in preparation).

Accepted for publication October 16, 1995.

Yehonathan Hazony *Boston University, Department of Manufacturing Engineering, College of Engineering, 44 Cummington Street, Boston, Massachusetts 02215.* Dr. Hazony is a professor of manufacturing engineering at Boston University. He obtained his Ph.D. in physics in 1965. His main research interest is in software automation for the design and implementation of engineering systems, with an emphasis on manufacturing.

Reprint Order No. G321-5595.