# Customized systems for engineering applications

by Y. Hazony
L. Zeidner

***An APL2™-based high-productivity software-development environment is shown to enable small teams of two or three persons to build complex engineering software systems. The productivity and flexibility of such small teams, equipped with this environment, enables them to build customized engineering application systems economically. These customized systems are far more useful for the particular applications they address than are the generic systems that are commonly produced by large software-development groups. A customized engineering application system is described, illustrating the productivity of the two APL2-based computer-aided software engineering (CASE) tools used for its implementation and long-term software maintenance. The system is presented in some detail, to demonstrate its sophistication and thus provide a measure of the productivity of the software-development environment. The two CASE tools that comprise this software-development environment are used to build interactive graphical application systems, and to build systems for applications that require or can benefit from distributed cooperative processing. A list of some customized application systems built using the described environment is provided, along with estimates of the implementation efforts. The features of APL2 that play a key role in the effectiveness of these tools are also discussed.***

Engineering software has traditionally been developed by large groups and its development has often been costly, slow, and inflexible. To distribute the cost, marketing strategy has called for developing large generic systems intended for broad markets over long product life spans. This paper presents two APL2*-based computer-aided software engineering (CASE) tools that enable a small software-development team of two or three persons to build customized application systems at low cost, in a timely manner, while remaining flexible to changes in requirements. This changes the marketing strategy so that narrowly-focused engineering application markets and short-lived problem domains can now be addressed effectively.

The complex engineering application described in this paper demonstrates the capabilities of the customized approach, and the power of the software-development tools employed in the implementation process. The application involves "seamless" processing from design to manufacturing, geometric modeling, and automatic process generation for 5-axis milling (three positional and two orientational degrees of freedom).[1–5] A qualitative comparison of this recent project and the estimated cost of development of comparable commercial systems suggests a reduction of one order of magnitude in the software-development costs.

One of the two CASE tools discussed is the Expert System Generator (ESG).[6,7] A customized engineering application system is an interactive environment that uses captured expert knowledge to help an engineer solve problems. The ESG separates the two main forms of complexity involved in creating a customized application system. One

form of complexity is the bottom-up development of each algorithmic component. The other is the top-down interconnection of these components and the associated interactive interfaces, into a useful integrated application system. The interactive interfaces enable the engineer to participate in the solution process where algorithmic solutions are not suitable or desirable, or have not yet emerged. The ESG enables the software-development team to use graphical programming to specify the top-down interconnectivity, flow, and interfaces, and relies upon APL2 as an efficient tool for expressing the algorithmic functionality of each component. APL2 also allows access to necessary algorithmic functionality implemented in other languages, by its interface to associated processors.[8] Together, ESG and APL2 provide an environment in which the design and implementation of a complicated customized application system requires roughly one person-year of effort.

The second CASE tool discussed is the Server Network Generator (SNG).[9–13] The SNG addresses applications that require or can benefit from distributed cooperative processing. The computer-controlled operation of a flexible manufacturing system is one such application. However, distributed cooperative processing (abbreviated DCP in this paper) is readily applicable to many other applications that exhibit either concurrency or parallelism. The SNG automatically converts a graphical specification of the application into an operational DCP network skeleton.

Application systems may be built directly using the combined SNG, ESG, and APL2 environment, or existing ESG application systems can be extended to benefit from distributed cooperative processing by using the SNG. Once the application's inherent concurrency is identified, a server network is designed graphically, distributing the application system's algorithmic components across a set of interconnected servers. Individual algorithms exhibiting concurrency can be decomposed and distributed across a set of servers, to be processed concurrently.

During the past several years, the various generations of the ESG environment have been used to create many customized engineering application systems (see Table 1). Some of these projects produced instructional systems for undergraduate and graduate classroom applications. Others were developed as part of industrial research contracts.

This paper begins with a description of a customized engineering application system, built using the combined SNG, ESG, and APL2 environment. Two of the system's algorithms and a related data structure have been chosen to illustrate the power of APL2 as the implementation language for bottom-up software development. The planar-chopping algorithm described is a direct translation of a mathematical concept into a simple APL2 expression. The SURROUND algorithm is more complex, with several components that present opportunities for DCP. APL2 enables the software developer to capture the SURROUND algorithm's inherent concurrency naturally, through the use of nested arrays, whether or not DCP is involved. When DCP is applied, its implementation is greatly simplified by this simple preparation. Next, the ESG and SNG are described briefly, along with a discussion of APL2 features that either were instrumental in their development, or are particularly advantageous in their use. Finally, productivity considerations are discussed.

## A seamless design-to-manufacture system

This customized application system is an alternative to the traditional computer-aided design/computer-aided manufacturing (CAD/CAM) approach to manufacturing. It overcomes the major disadvantage of CAD/CAM systems, the many "seams" between separate interactive components, which are opportunities for introduction of human errors and are obstacles to flexible rapid prototyping. In this seamless design-to-manufacture (SDTM) system, changes made to the part design can automatically result in a new prototype rapidly, without the time-consuming involvement of several disjoint interactive CAD/CAM steps. Part and process design changes are performed interactively at the conceptual level. The resulting geometric repercussions have been completely automated, so that applying varying levels of computing hardware power can yield corresponding levels of rapid-prototyping speed. This presents an opportunity for the introduction of distributed cooperative processing, as will be shown.

This SDTM system has been demonstrated and is undergoing further development. It represents an estimated effort of 1.5 person-years, and will have

**Table 1 A list of implemented customized application systems**

| Expert System | Person-Years | Date |
|---|---|---|
| Hybrid LSI Circuit Design System (in collaboration with A. Mavretic and I. Tham) | 2 | 1986 |
| Conceptual Aircraft Design System—An instructional system (in collaboration with G. Succi and J. O'Brien) | 1 | 1986 |
| The Expert System Generator (in collaboration with A. C. Williams) | 2 | 1986 |
| Propeller-Blade Design and Manufacture (in collaboration with G. Succi and W. Ruiz) | 0.5 | 1987 |
| Control Systems Design—An instructional system (in collaboration with H. D'Angelo and W. Curtin) | 2 | 1987 |
| Design and Manufacture of Jet-Engine Combustion-Chamber Lining (in collaboration with T. Shojaie and W. Ruiz) | 1 | 1988 |
| The BUMES System—A design system for NC turning machines (in collaboration with T. Shojaie and S. Sadri) | 1 | 1988 |
| The Server Network Generator (in collaboration with S. Bernstein, Z. Nour) | 1 | 1989 |
| Automatic Process Generator for 3-Axis Milling | 0.5 | 1989 |
| SEPTOR Project—Control of Manufacturing Transfer Lines (in collaboration with E. Ebner, S. Rastogi, and A. Tuczapec) | 0.5 | 1990 |
| Geometric Modeler for 5-Axis Milling | 1 | 1990 |

required roughly twice as much effort when completed. The application is the design-to-manufacture of sets of customized golf club heads (see Figure 1). The complexity of this engineering application is due to its sophisticated geometric definition, its family of parts in excess of one hundred members, the mechanical and aerodynamic considerations, and the overriding necessity of manufacturability and ease of rapid prototyping.

The concepts and system components, discussed here in this context, apply to a wide range of products requiring the sophistication of 5-axis milling (three positional and two orientational degrees of freedom), and can be generalized to apply to other manufacturing processes. The geometric modeler for 5-axis milling, and the automatic process generator for 3-axis (or three positional degrees of freedom) milling[2,4] (see also Table 1), are components of this SDTM system.

An appropriately constructed geometric model of the part, stock material, and fixtures, combined with a complete set of process rules and data, provides all the information necessary for the implementation of automatic process generation. In the absence of some of the necessary rules or data, an SDTM system must provide for direct interactive input from a human expert in guiding the process at the conceptual level. Examples of such missing information, supplied by the human expert, are the choice of major process orientation and determination of the fixture required to hold the workpiece.

The overall structure of the customized SDTM system uses the geometric model generated in the part-design stage, and the specifications of the machining center and tooling, to produce numerous auxiliary geometries automatically, in the process of converting the designed part into a manufactured product. Figure 2 illustrates the flow of the design process generating the golf club head illustrated in Figure 1. The diagram depicts the various design stages and the evolving geometric model. Figure 2 serves to illustrate the

structural complexity of customized application systems that can be developed with the ESG environment.

**Geometry generators.** The SDTM system's *geometric modeler* is able to guarantee the geometric integrity [1,3,5] of the part model due to the geometry generation process. The geometric modeler is able to represent objects that can be described in terms of the intersection, union and subtraction of convex objects. It provides a simple solution to the problem of geometric integrity for large and diverse families of convex and concave geometric designs.

The geometric modeler represents convex objects internally as a set of planar polygonal facets. [1,3,5] Each of the part's constituent convex objects corresponds to a part feature. Within the SDTM system, each feature and its connection with the rest of the part are described parametrically. Each type of feature is associated with a *geometry generator* that uses these parameters to generate the feature's faceted model automatically. The ESG enables the software developer to implement the feature-based parametric specification interfaces using graphical programming, and provides a structural framework within which the automatic geometry-generation software fits.

The geometric modeler combines the advantages of a broad range of acceptable mathematical surface representations (and hybrids) at the conceptual and parametric part-design level, with the computational advantages of planar faceted representation at the process-geometry level. The Boolean-geometry processor used to construct the object from its convex components using intersection, union, and subtraction, operates on the faceted model. At the process-generation stage, most systems are forced to convert whatever mathematical surface representation they employ to linear and circular elements, to conform to the primitive motions available using numerically-controlled machining equipment. The geometric modeler merely converts immediately to simplify its own internal computation, while providing parametric control of the surface smoothness.

*The family-of-parts geometry generator.* A *family-of-parts* shares sufficient similarity that can be exploited to lower the cost and reduce the time required to design and manufacture successive family members. The SDTM system was intended specifically to exploit the similarity in design and manufacture of a particular brand of semicustomized golf club heads. This particular family-of-parts is partially defined by a combination of three discrete design parameters, combining to describe over one hundred family members:

1. Loft angle—the angle between the ground and the sloped club face that hits the golf ball
2. Lie angle—the angle between the ground and the club shaft held by the golfer
3. Swing weight—the weight of the golf club head, controlled by the depth of the back pocket

Additional geometric parameters are needed to fully characterize this family-of-parts. The SDTM system enables the part designer to specify all of these parameters. The family-of-parts geometry generator converts these parameters to a fully-defined planar-faceted geometric model, as shown in Figure 1.

Figure 2 shows the constituent geometry generators that pertain to specific features of the club head. Each of these generates convex objects that are combined by Boolean geometric operations to generate the complete club-head geometry. Figure 2 demonstrates the inherent concurrency in this computation. It presents an opportunity to benefit from the computational advantages of distributed cooperative processing through the use of the combined SNG, ESG, and APL2 software-development environment.

**Figure 1** A geometric model of the head of a golf club. The model is generated using the geometric modeling system described in the text.

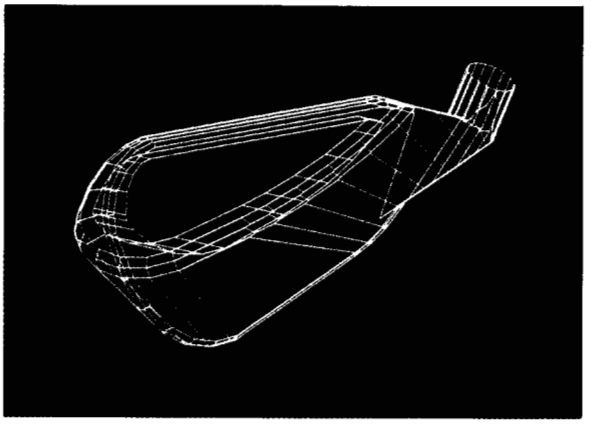


**Figure 2 Part and process design flow**

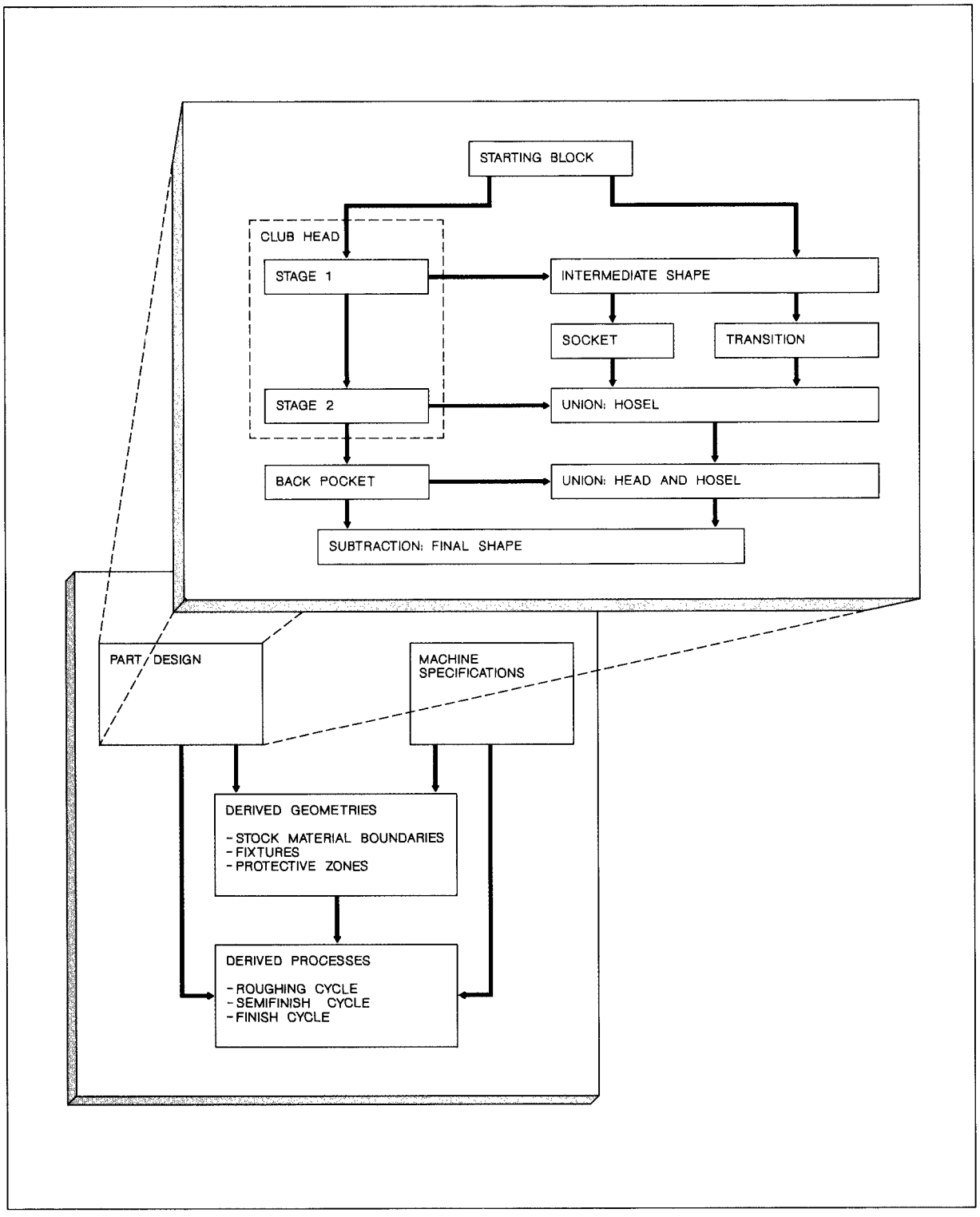
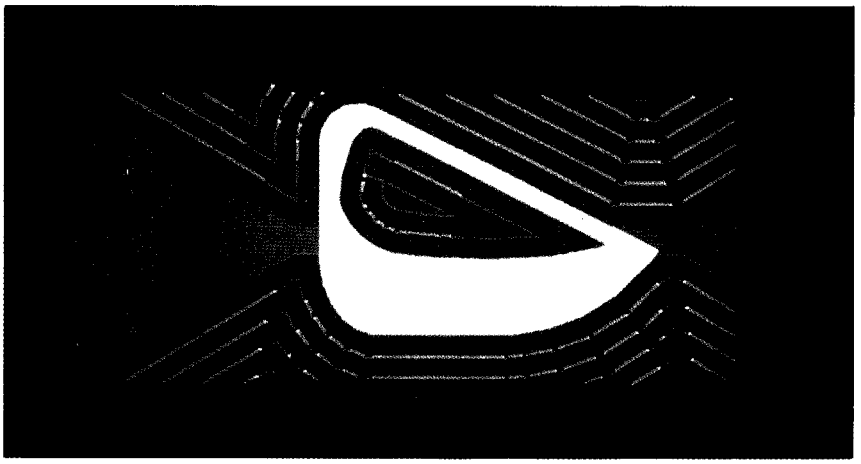


**Figure 3 Automatic process generation for one of the layers shown in Figure 6**



*Manufacturing process geometry generators.* The manufacturing process for machining this family-of-parts is implemented in the following steps:

1. The rough-milling process
2. The semifinish-milling process
3. The finish-milling process

For each process, the manufacturing process geometry generators constitute a set of algorithms producing all the details associated with:

- Defining the tool approach
- Creating a continuous trajectory to remove material according to a specific process-governed strategy
- Subdividing the trajectory into sections requiring different feed rates
- Defining the tool retraction

Figure 3 provides an example of such a complex trajectory associated with the 3-axis milling operation. Figure 4 illustrates some of the geometric problems that must be addressed to produce such a trajectory automatically.

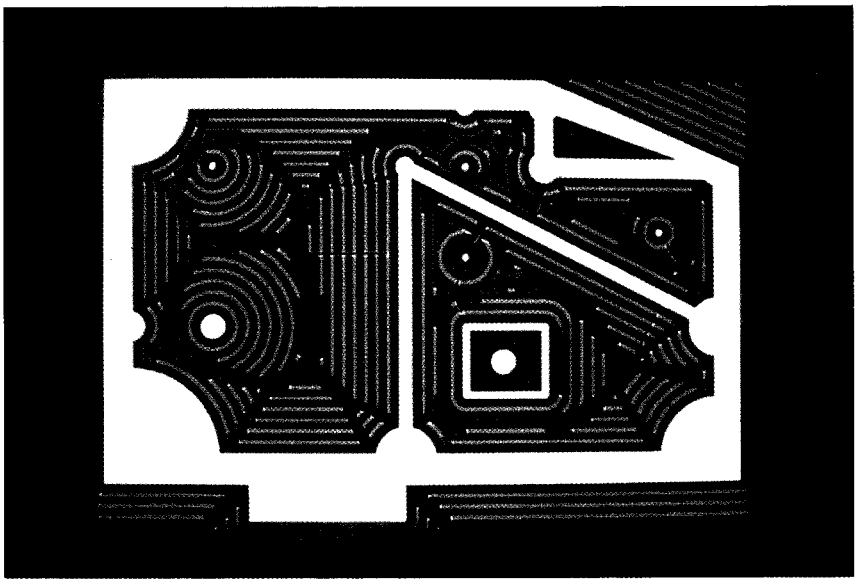
The flow diagram shown in Figure 2 represents only those geometry generators used to produce the part geometry. The part geometry must be combined with stock material geometry and fixture geometry as input to the automatic process generation software. Figure 5 represents all of these geometries.

Three manufacturing process geometry generators are described next. The first creates auxiliary geometry as input to the next two. All three generators were implemented in APL2 with the data-structure flexibility provided by nested arrays.

*The slice geometry generator.* The 3-axis rough-milling process employs a rotating cylindrical milling tool that is lowered so that its bottom end is a specific distance (rough depth of cut) below the surface of the material. The tool is then moved laterally so that its cylindrical milling surface removes material in its path. The material opposes the tool's forward movement, thus deflecting the tool, causing its actual travel path to differ from the desired path. The process planner varies the rough depth of cut (RDC) parameter within the SDTM system's conceptual process-design facility. Based on the type of material, he varies the RDC parameter in conjunction with the feed rate at which the tool progresses laterally, and the tool radius, so that the deflection will be acceptably small. The ESG simplifies the task of graphically designing and implementing the process-design facility, and provides the structural framework into which the rules governing acceptable choices and combinations of process parameters fit.

The rough-milling process is thus performed at a sequence of progressively deeper layers, each

Figure 4 Application of the SURROUND algorithm to a part design with multiple pockets and islands within pockets



deeper into the stock material, by an additional distance of RDC, each exposing the next new stock material surface. A slice geometry generator is used to determine the part, stock material, and fixture geometry at each rough-milling layer. This generator intersects the complete geometry with a set of parallel equidistant slicing planes, separated by a distance of RDC from one another. The result is a set of slices, each consisting of some number of part loops, some number of stock loops, and some number of fixture loops, as shown in Figure 6.

*The 3-axis rough-milling process-geometry generator.* The 3-axis milling process is performed sequentially slice by slice. The automatic process-generation software works on each slice independently, thus providing an opportunity for

Figure 5 The combined model of the geometries of the part, the stock material, the fixtures, and the protective zones

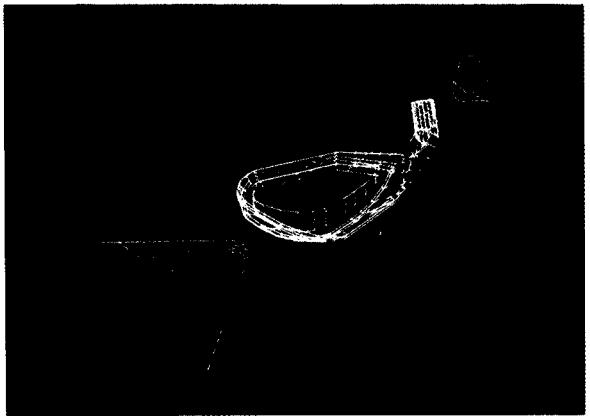
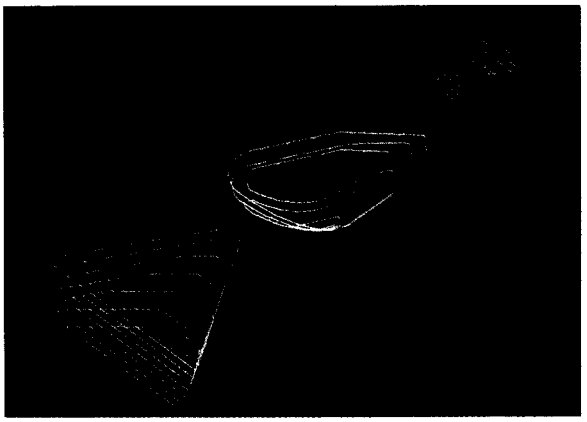


Figure 6 Equidistant slices through the combined geometric model of the part, stock material, fixtures, and protective zones



concurrent processing. The rough-milling process requires the tool to be guided along a path with its axis of rotation orthogonal to the slice plane. At each slice depth, as the tool follows the path, it removes the majority of stock material that is outside of the part and fixture geometry loops. A small amount of material is intentionally left immediately surrounding the part and fixture geometry. This material is removed later, during the semifinish and finish processes.

The goal of the rough-milling process is to remove material as quickly as possible. This process is performed with a large-diameter tool moving quickly and is designed to tolerate inaccuracy caused as the tool is deflected by the enormous forces exerted upon it. Much less material is left to be removed by the semifinish and finish processes, thereby exerting far weaker forces upon the tool and deflecting it less. Each successive process removes material closer to the part geometry and must therefore be more accurate. The semifinish process smoothes out the remaining material left by the rough-milling process. If the layer were not smoothed to homogeneous thickness the tool would confront widely varying deflection forces during the finish process and would be unable to maintain the necessary accuracy.

During the rough-milling process, the tool must often remove a quantity of material that is wider than its diameter. It therefore proceeds to take several passes, each digging laterally further into the material, taking advantage of the accessibility provided by the previous pass and providing further accessibility for the next pass. The width of each pass is a process parameter that depends upon the material, the power of the milling machine, and the RDC, but which cannot exceed the tool's diameter.

Each pass is thus intended to expose a specific geometric surface that will become the stock material geometry that it leaves behind. The automatic process-generation software works backward from the part geometry and fixture loops to compute the successive passes required for each slice. Each pass consists of a path along which the center of the tool bottom must move. Computing this set of tool paths based upon the slice loops involves an analytic geometry problem referred to as the SURROUND problem.[2,4] Figure 3 shows a part geometry and the concentric loops generated by the SURROUND algorithm.

These loops, when traversed at appropriate feed rates, in accordance with manufacturing process strategy, constitute a tool path for one slice. The manufacturing strategy is intended to balance the goals of minimizing the total machining time and milling as accurately as possible. This strategy is implemented in a TRAVERSAL algorithm, which converts a set of concentric SURROUND loops into a tool path consisting of geometry and feed rates.[2,4] The feed rates ensure that the tool moves slowly when there is greater force opposing its travel, and thus greater deflection, faster when there is less opposing force and deflection, and fastest when it is not in contact with any stock material. The feed rates are computed during the TRAVERSAL algorithm, based upon the choices it makes in traversing the SURROUND loops. The feed rates change continuously along the tool paths based upon these traversal choices. The SURROUND and TRAVERSAL algorithms each present opportunities for employing distributed cooperative processing, using the combined SNG, ESG, and APL2 environment, due to their inherent concurrency, as is explained next.

As the milling tool follows a tool path to expose a desired surface, its circular cutting edge is unable to reach into concave inside corners in the surface. This creates a disparity between the desired surface and the resulting stock material surface. The SURROUND algorithm is used to compute the resulting stock surface, by SURROUNDing the tool path geometry at a distance of one tool radius. This provides the stock geometry prior to the semifinish and finish-milling processes.

*The 5-axis finish-milling process-geometry generator.* The 5-axis finish-milling process is performed by rotating the part about its X-axis (using a rotational control axis defined as *A*), while positioning the tool so that its end is touching the desired part-geometry surface, and orienting it so that the tool's rotational axis is orthogonal to the local surface of the part. A slice geometry generator is used to generate a set of equidistant slices parallel to the Y-Z plane. Each slice consists of one part-geometry loop.

Figure 7 shows the Y-Z motion of the tip of the tool as it traverses along one part-geometry loop on a slice. The X coordinate is constant for a slice, and two orientational axes, A and B, are continuously changing. (Only axis A is shown in the

Figure 7 The Y-Z trajectory of the tip of the milling tool, corresponding to the finish cut of one slice

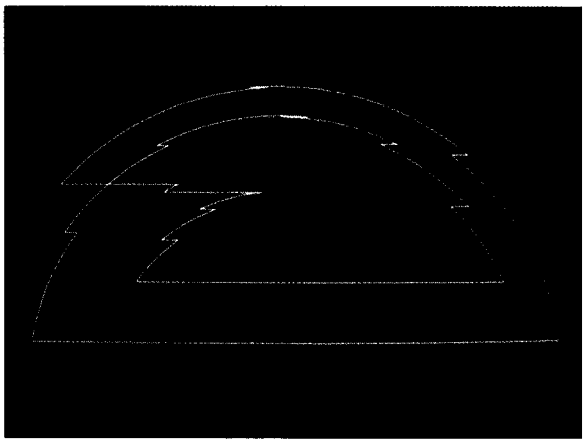


Figure 7 example.) The complexity of the Y-Z trajectory is due to the geometric coupling between the A, Y, and Z axes, which is characteristic of the geometry of the particular milling machine. The actual contour being generated by the motion described in Figure 7 is represented by a relatively simple convex polygon, which is an ordinary cross section through the part geometry.

**A data structure for geometric modeling.** To achieve the goals of SDTM, the geometric modeler must describe the full geometry of the problem, including part, stock, and fixture geometry. These combined data are necessary for generating the process geometries employed in the automatic generation of numerical control software, which drives the numerical control machines for part manufacture.

Figure 8A depicts a depth-5 nested vector used to describe the full geometry of the problem.[3] The vector consists of three depth-4 elements, describing stock, fixture, and part geometry. Each geometric element is defined by a data structure (Figure 8B), consisting of a plane table (PT), a node table (NT), a link table (LT), and a nested facet vector (FV). Each row of PT, NT, and LT corresponds to a plane, a node, and a link, respectively. Each item of FV corresponds to a facet, and consists of a link-number table, a plane number, and other attributes.

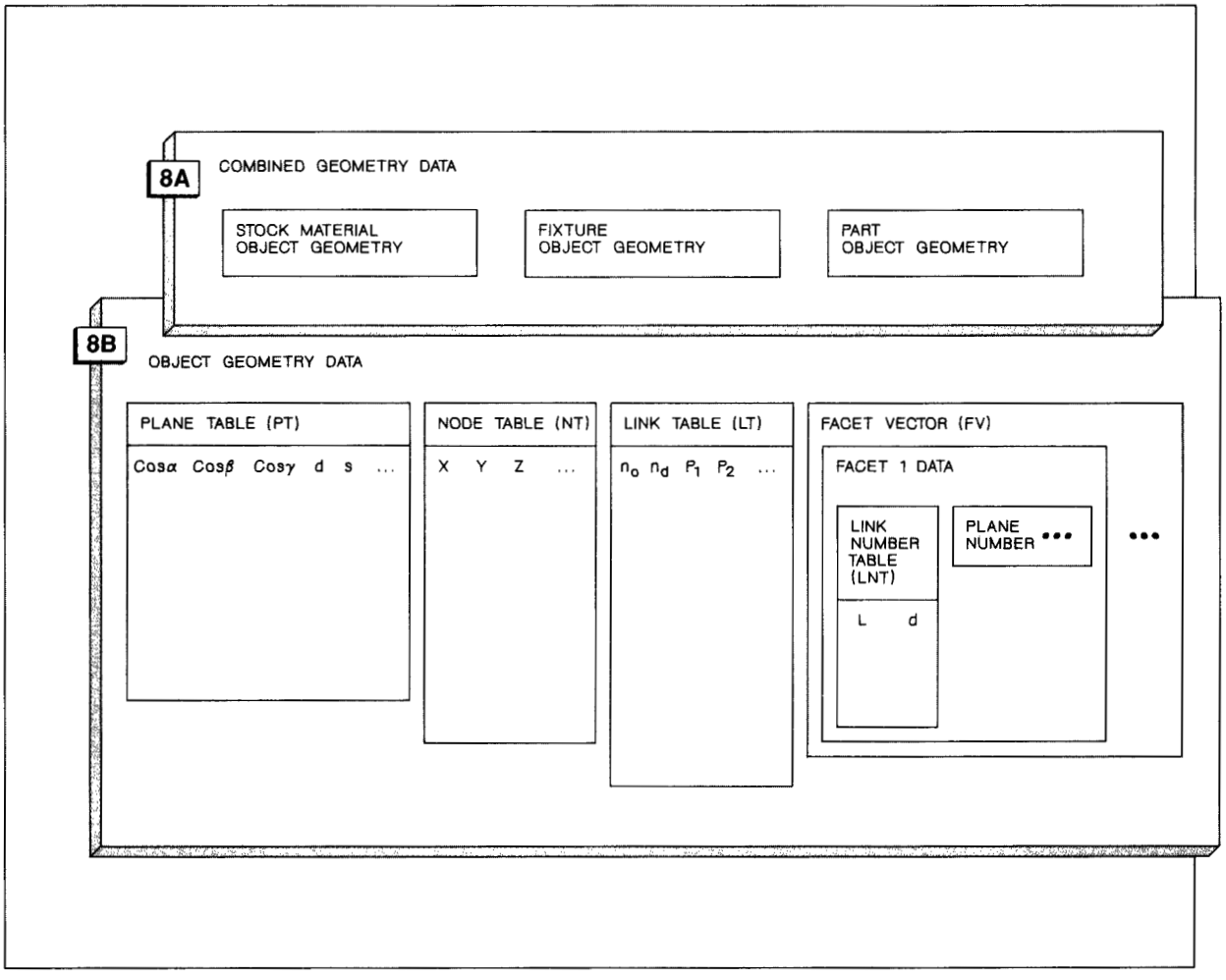
An arbitrary geometric shape may be described in terms of a simple start-up geometry and a collection of geometric features attached to it. The implementation of a fully automated SDTM system requires a full parametrization of the start-up geometry, all of the attached features, and their geometric interrelations. While a wide range of geometric features are amenable to direct mathematical parametrization, many shapes are designed in an ad hoc fashion, with no obvious approach to design parametrization. The start-up geometry generator provides a means to construct an object, or a feature, in terms of an arbitrary but complete set of facets. In the absence of any known geometric interrelations between the various planes, the initial parametric definition of the start-up geometry consists of the parametric definition of the generating planes.

The completeness of the set of facets is ensured by applying a planar chopping operation to successively "chop off" parts of an initial valid oversized shape. The set is complete if none of the initial object's facet planes remain. Each chopping operation may eliminate some facets and modify some of the facets comprising the previously-defined valid geometry. Since a facet is defined by a polygon in a given plane, the modification of a facet may consist of deletion of some links in the polygon, shortening of some links, and the creation of new links. The collection of all new links created in one planar chopping operation define one or more new facets lying in the chopping plane.

A clockwise polygonal definition is used to indicate which of the two three-dimensional spaces bounded by the facet plane is solid material. Since each link is shared by two adjacent facets, clockwise traversal of the two polygonal facets traverses the link in opposite directions. The rows of the link number table correspond to the links of the facet polygon, the first column containing pointers to links in LT and the second column indicating the direction in which each link is traversed while proceeding clockwise around the polygon.

To maintain geometric integrity, no redundant nodes, links, or planes are permitted in PT, NT, or LT.[1,3,5] The first two columns of LT contain pointers to the node table NT for each link's origin and destination nodes. The third and fourth columns contain pointers to rows of the plane table PT, denoting pairs of planes, which define pairs of

**Figure 8 Data representation of the combined geometry**



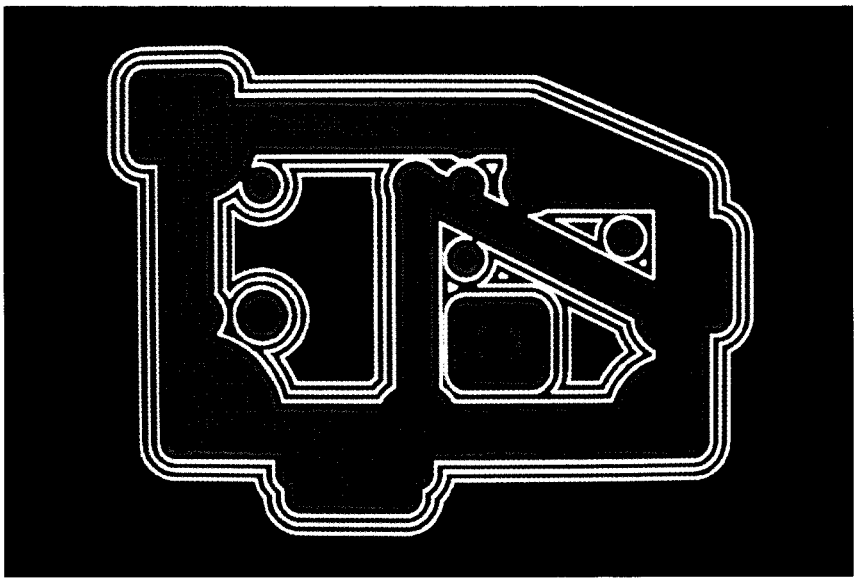

facets that share each link. Attributes may be added as additional columns in PT, NT, and LT.

The significance of using a simple mechanism to maintain an attribute structure is illustrated during the manufacturing process generation. The plane attributes included in the third and fourth columns of LT are used in the generation of 5-axis interpolated tool trajectories for the finish process, as described next.

This data structure is one example illustrating the power of nested arrays as implemented in APL2. The ease with which such a data structure can be devised and implemented is one of the simplifying, time-saving tools that facilitates the implementation of application systems.

**An implementation of the planar chopping algorithm.** The planar chopping algorithm is fundamental to the implementation of planar-faceted object geometry. It takes an object geometry and a chopping plane, and keeps the portion of the object geometry that lies on one side of the plane. A convex object is constructed by beginning with an initial oversized valid solid geometry, and successively applying the chopping algorithm, plane-by-plane. Each chopping operation separates the object's nodes into two groups, each lying on opposite sides of the chopping plane, and removes one of the groups of nodes.

**Figure 9 The concentric SURROUND loops generated from a part boundary**



Planes are defined in PT in terms of their normal representation. The first three columns contain the three-dimensional direction cosines ($\alpha$, $\beta$, $\gamma$) of the normal to the plane. The fourth column contains the distance ($d$) of the plane from the origin, and the fifth column contains the *side parameter*, a 1 or $-1$ denoting which side of the plane contains the solid object. Additional columns are used for various attributes.

The following algebraic equation represents the distance $D$ of a point ($x$, $y$, $z$) from a plane:

$$D = x \cos \alpha + y \cos \beta + z \cos \gamma - d$$

where the plane is defined by the three direction cosines and the distance parameter $d$, described above. This formula is applied to all of the nodes in the object to determine which side of the plane they are on. The classification is performed by the following APL2 expression:

```
0 > S × D
```

in which $S$ is the side parameter defined above. This expression produces a Boolean vector that is used to compress the node table and eliminate all nodes on one side of the chopping plane. Given the node-coordinate matrix *NT*[;⍳3], and the five-element vector $P$, which defines the chopping plane in the same format as each row of the plane table PT, the APL2 expressions:

```
(NT[;⍳3],¯1) +.× P[⍳4]
(0 > P[5] × D)/[1]NT
```

removes all points on one side of the chopping plane.[3]

These APL2 expressions combine the algebraic equation with the Boolean operation to produce the desired data. The SDTM system includes numerous instances of algorithmic solutions. Many of these are as direct a translation of the corresponding mathematics as this example. This productivity of direct translation from mathematics to APL2 software implementation encourages the software developer to use powerful mathematical formulations whenever possible.

**An implementation of the SURROUND algorithm.** Figure 9 illustrates several instances of application of the SURROUND algorithm. The solid blue geometry loops were SURROUNDed, resulting in the red loops. The remaining yellow concentric sets of loops were obtained by iteratively SURROUNDing first the red loops, then the resulting yellow loops, and so on, sequentially. Alternately, the yellow concentric sets of loops could

have each been computed independently by SURROUNDing the solid blue loops, at various distances. This independent approach presents an opportunity for distributed cooperative processing by computing all or some of the yellow sets of loops concurrently.

SURROUNDing an individual convex loop, on the outside, at a distance of $d$, is relatively straightforward. For each line segment in the loop, it consists of generating a line parallel to it at a distance $d$. For each arc in the loop another concentric arc must be generated at a radius greater by $d$. The sequence of the new loop's elements is identical to those of the original loop.

SURROUNDing a set of loops, each consisting of convex and concave geometry, is more complex. Concavity and the existence of multiple loops can introduce regions where the tool cannot fit. The resulting SURROUND loops may not correspond directly with the original loops. Some elements of the original loops may have no corresponding elements in the SURROUND loops, while other elements of the original loops may correspond to several separate elements in the SURROUND loops.

SURROUNDing a set of loops can be formulated either as a sequential or concurrent algorithm. The concurrent SURROUND algorithm consists of five steps, four of which exhibit concurrency. The five steps are: candidate generation, finding intersections, finding midpoints, classification, and threading.

*Candidate generation.* Candidates are all of the line segments and arcs that are needed to create the SURROUND loops, although each candidate may or may not be used in whole or in part. Each candidate can be generated independently of all others. This presents an opportunity for distributed cooperative processing (DCP) with a high degree of flexibility because the number of candidates to be generated concurrently can vary based upon the number of available concurrent processors.

*Intersections.* All intersections between candidate segments must be identified. This is an arc and line segment intersection problem, involving linear and angular range calculations. It too can be computed concurrently. Each concurrent task is to determine all intersections between two subsets of the set of candidates. The two subsets may or may not have elements in common. This step presents a high degree of flexibility for DCP because the choices of subset sizes and the number of subsets can vary based upon the number of available concurrent processors.

*Midpoints and classification.* Next, midpoints are identified along the candidate elements, between consecutive intersection points. All of the intersection points and midpoints must be tested to determine their closest distance to the original set of loops. Since they lie on candidates, they are at most distance $d$ from the original loops. However, due to concavity and to the existence of multiple loops, they may actually be on or inside of another loop. Each point is classified as being either (1) inside or on the original set of loops, (2) closer than distance $d$ outside the original set of loops, or (3) at precisely distance $d$ or farther from any point on the original set of loops. Each candidate is then subdivided, based upon the intersection points and midpoints. Only those segments of the candidate elements between two points of the third classification, above, are kept.

Midpoint computations can be computed independently for each candidate. This provides the same sort of DCP flexibility as candidate generation. Classification can be performed concurrently by comparing sets of points with sets of original elements, with the same DCP flexibility as finding intersections.

*Threading.* The candidate segments that remain after classification constitute the complete geometry of the SURROUND loops. However, they must be resequenced and subdivided into separate loops. Threading is performed by matching endpoints and is the least computationally intensive step. The matching could be performed concurrently, but there is little incentive.

By analyzing detailed timing results, using APL2's supplied workspace, it was possible to identify portions of the SURROUND algorithm that consumed the most time. A few of the lowest-level routines were identified and recoded in FORTRAN, and then linked into the SURROUND algorithm by using APL2's associated processor interface. This selective recoding effort improved the overall algorithmic speed significantly, while involving only a very small portion of the SURROUND software.

## The Expert System Generator

Engineering application systems, built using the ESG, consist of algorithmic engineering software components bound together by a control structure. The software-development team specifies the top-down control structure using the ESG. Through automatic code generation, the ESG creates an operational system skeleton in which all control functions can be exercised, even before the bottom-up implementation of the engineering algorithms. The ESG generates empty APL2 functions as placeholders into which the specific algorithm will be implemented in a bottom-up process. The application system is designed as a set of facilities, or separate environments, each of which focuses the available computational and interactive resources on a specific portion of the overall engineering problem. The ESG accommodates any structural network of facility interconnection that is deemed appropriate by the development team.

Engineering rules are critical to the implementation of customized application systems. The ESG provides well-defined mechanisms with which to implement these rules, which are applied to data entered either textually or graphically, and are used to regulate the transfer of control between facilities.

Throughout the development process, a working prototype system is readily available for testing. The control structure can be changed graphically at any point in the development process, resulting in automatic code maintenance, and thus an appropriately modified system. Typically, the software-development team begins with a simple control structure and adds algorithmic functionality until enhancement of the control structure is necessary. The development process alternates between augmenting the control structure graphically, augmenting the engineering algorithmic components, and testing the prototype. The prototype is the application system.

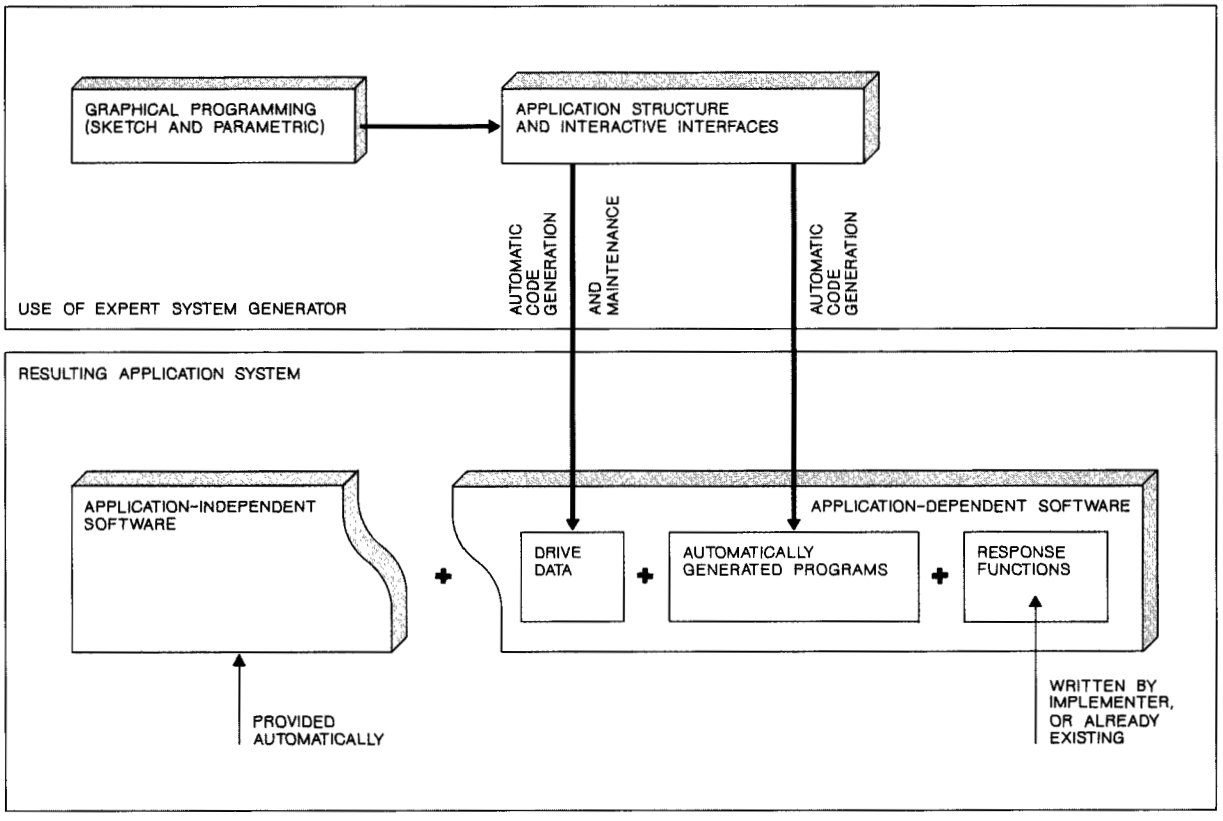
The development of customized engineering application systems involves learning all of the various types of application rules and understanding the relationships between the many sources and uses of data. Although engineering application experts exist, their areas of expertise may be more localized than the desired application system, their understanding of this information may not be organized in an appropriate form for automation, and they may not be expert at articulating this information to a software developer. Learning the engineering application is an incremental process that is accelerated by the discussions and realizations that arise during frequent review of system prototypes. Flexibility is crucial during this process, and an important measure of flexibility is the software developer's willingness to abandon an approach and adopt a new one. The productivity of the ESG reduces the cost of rapid prototyping to a level that encourages flexibility. While most software-development systems are judged on their ability to easily represent the final software implementation, the ESG environment also excels at providing structure and flexibility during this necessary learning process, so that it is accelerated. Graphical programming, automatic code generation, and maintenance in the ESG, and easily modifiable nested arrays in APL2 provide this power.

Figure 10 illustrates the relationship between the use of the ESG and the resulting application system. The top of this figure shows the graphical and parametric programming that takes place within the ESG to capture the control structure and interactive interface of the application system. This information is automatically code-generated, modifying the system skeleton. The bottom of the figure represents the resulting application system, which consists of four components discussed in the following paragraphs.

**Application-independent software.** Although different customized application systems consist of different control structures and interactive interfaces, these structures and interfaces can all be implemented in terms of a small finite set of software. This software is analogous to a microcoded processor. The functionality of this generic software is customized, for each application system, by the drive data (described next).

**Drive data.** The drive data are automatically code-generated by the ESG. Constituting all of the application-dependent control structure and interactive interface information gleaned from the graphical programming done in the ESG, the drive data, as an entity, are analogous to microcode which, together with a microcoded processor, provide customized functionality.

**Figure 10** Use of the Expert System Generator causes automatic code generation of two of the four software components of the resulting application system

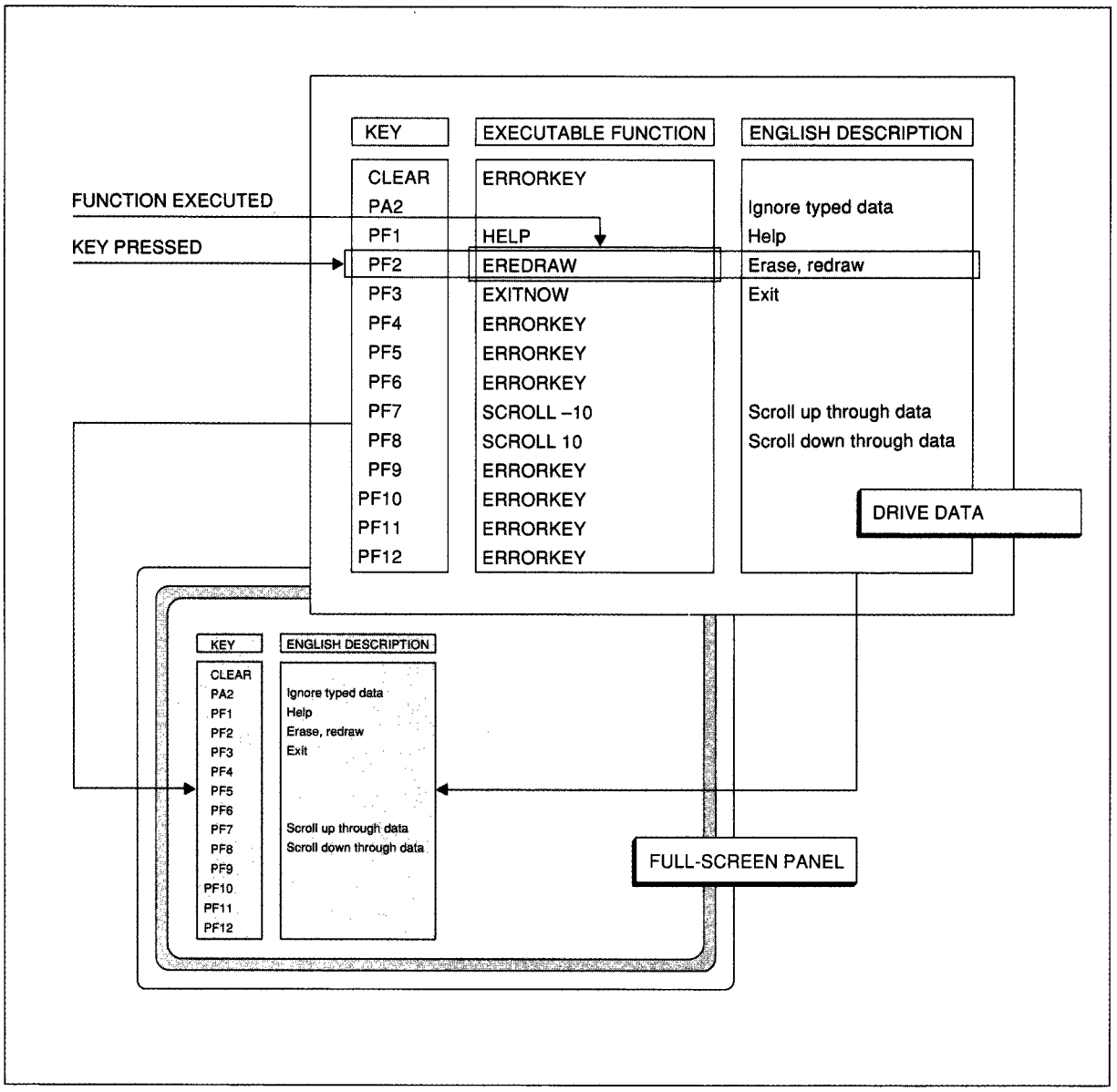



**Automatically generated programs.** These programs are a mechanism that permits the software-developer to provide capabilities that exceed those offered by the current version of ESG. They ensure that the extended functionality does not interfere with the normal operation of the application-independent software.

**Response functions.** Response functions are the engineering algorithmic components of the application system. They are executed by the application-independent software, as directed by the drive data, in response to interactive actions by the engineer. Typical response functions include those invoked as a result of pressing program function keys, and those invoked as a result of typing data into the fields of a full-screen panel.

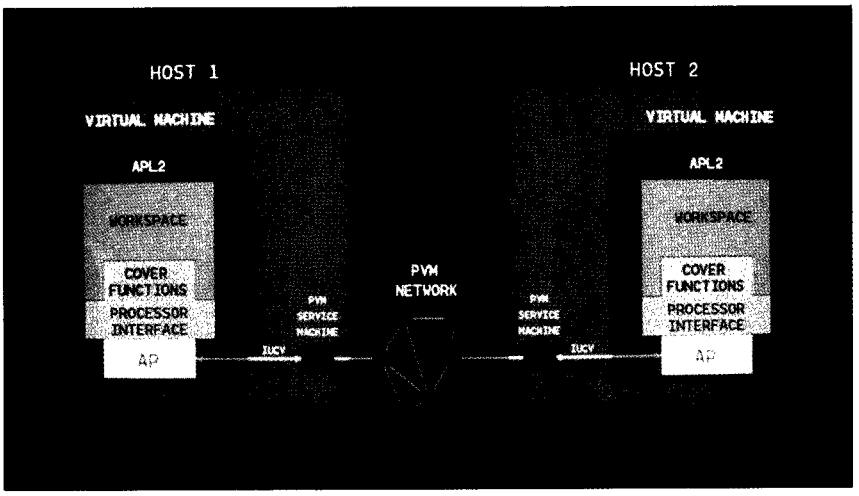
The ESG uses the capability of APL2 to treat data as executable software. Figure 11 illustrates a drive-data array containing a selection of program function keys, a list of expressions or response functions to be executed when the corresponding key is pressed, and a list of explanatory key definitions. When the application system is used, the key names and their definitions are automatically written to the workstation's textual screen, providing the engineer with a description of available options. When the function key is pressed, the application system selects the corresponding response function from the drive data array, and executes it. This is the entire controlling mechanism: the drive data array actually "drives" the application. The contents of the drive data array are used by the application-independent software for:

**Figure 11 The use of data to "drive" the expert system**




- Controlling key definitions
- Writing data from the engineering database to the fields of the full-screen panel
- Reading data from the panel, checking the validity of the data, and writing the data back to appropriate locations within the database
- Moving from one facility in the application system to another
- Performing work between facilities
- Conditionally limiting passage between facilities
- Controlling a variety of complex application-dependent graphical cursors
- Automating retrieval and archiving of engineering design data
- On-line documentation

Figure 12 The implementation of the distributed interuser shared-variable interface




## The Server Network Generator

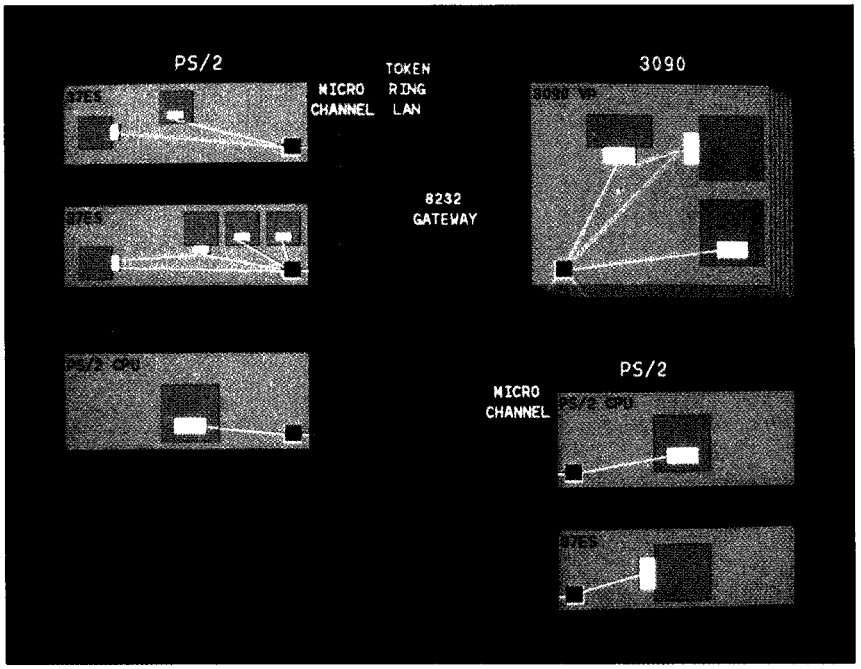
The seamless design-to-manufacture (SDTM) application system provides many examples of opportunities to benefit from distributed cooperative processing (DCP) due to concurrency inherent in various parts of the problem and the algorithms. Recent advances in computer and communication hardware technology have made networks of powerful computer resources affordable and commonplace. However, the effort required to develop an application that uses tight coupling of many networked processors has been high enough to restrict DCP primarily to research efforts and high-budget projects.

There are many obstacles to DCP that must be addressed, aside from the computing and communication hardware, if an application is to be recast as concurrent. Interprocess communication utilities exist but they vary in ease-of-use and performance. One must be chosen and integrated into the existing software. At the conceptual level, the application must be examined to identify the opportunities for concurrency, and to weigh the potential computational savings against the required software-development effort and the increased system complexity. The development of distributed software is fundamentally different from that of single-threaded software. It is frought with different logistical problems that stem from the processors' independence. The single-threaded software developer's main debugging strategy is to identify a situation that causes an error, reprogram, and then retry. This strategy is not available in DCP software development because of the many sources of timing variability in the separate processors and the intercommunication mechanisms. Even if the identical DCP programs are run again, they will not necessarily fail again at all, or in the same way, or at the same time. Other obstacles involve well-known problems regarding distributed access to data by software, version control, etc.

The combined SNG, ESG, and APL2 software-development environment overcomes many of the obstacles to DCP, so that it is relatively easy to create a tightly coupled DCP server network quickly. Furthermore, the flexibility of this environment makes it easy to adjust server networks to adapt to changing system requirements. A prototype of this combined environment is operational, and has been demonstrated and described in some detail.[9-13]

The conceptual interprocess communication vehicle used in the combined SNG, ESG, and APL2 environment is the APL2 interuser shared variable.[14,15] However, the current restriction of the APL2 shared variable implementation to a single host has been eliminated recently by the development of an auxiliary processor (AP).[9] Figure 12 illustrates this implementation. The AP is written

**Figure 13 The relationship between the distributed hardware platform and the distributed server network software platform**



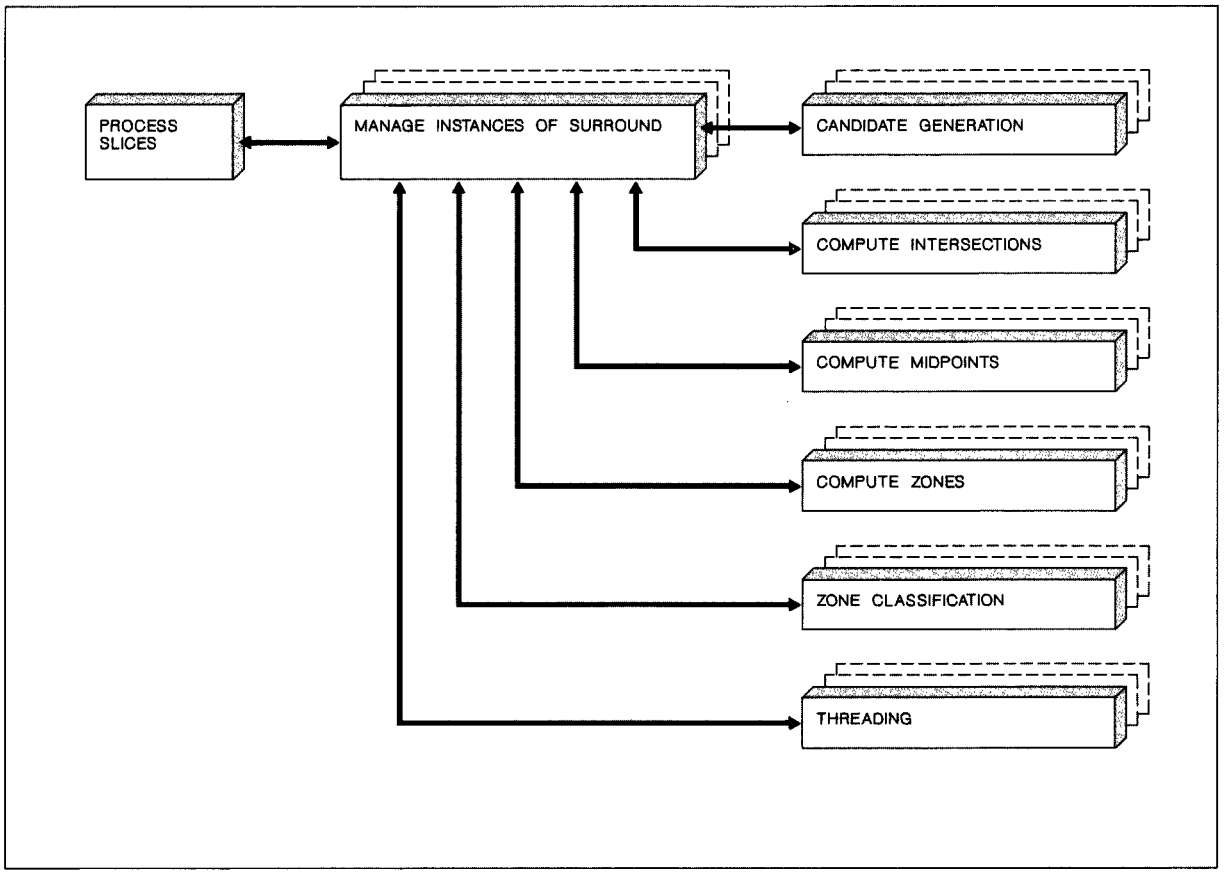

in assembly language and uses IBM's APL2 processor interface[16] to communicate with the APL2 workspace. The IBM Virtual Machine/System Product component, Inter-User Communication Vehicle (IUCV) communicates with the partner workspace in another host using the Virtual Machine/370 VM/Pass-Through Facility (PVM).[17] Various APL2 cover functions in the user's workspace establish communication with the AP, and substitute for the system □*SV* functions, with the same protocols and behavior.

Figure 13 illustrates the relationship between the distributed hardware platform and the distributed server network software platform. This research effort was developed on a network of IBM 7437 Virtual Machine/System Product Technical Workstations,[18] each consisting of an IBM System/370* processor attached to the microchannel of an IBM PS/2*. Each of these IBM 7437s is to be replaced by one or more IBM 37ES research-prototype System/370 processors, creating a configuration of token-ring networked clusters of microchannel-connected IBM 37ESs. Related research has demonstrated the ability to use IBM Clustered FORTRAN to distribute an application across two IBM Enterprise System/3090* multiprocessors, each with six Vector Facility features.[19]

Distributed interuser shared variables overcome one obstacle to DCP by providing high-level hooks to the application; however, many other obstacles remain. Using the SNG, the software developer graphically decomposes the application to express its inherent concurrency, and the data flow between the concurrent processes. This graphical programming automatically results in the top-down implementation of the DCP system skeleton. The software developer writes the response functions within each server that are automatically invoked by the system skeleton. As with the ESG environment, a prototype application system exists at all times during the development of server networks, and provides the catalyst to determine changes in requirements early in the development process.

Figure 14 shows a block diagram representing the automatic process generation SURROUND algorithm previously described, which lends itself to cooperative processing in many ways that can provide dramatic computational speedup when

Figure 14 A block diagram decomposition of the SURROUND algorithm




performed on a DCP platform.[18] Each slice can be processed independently. Each concentric SURROUND calculation can be performed concurrently, based upon the original innermost contour, using appropriate distance values. Within each SURROUND calculation, candidate generation can be partitioned into any number of concurrent computations, as can intersections, midpoints, and classification. Threading is sequential, but is not computationally intensive. Figure 14 represents a simplified diagram of an approach to decomposing this application. It shows duplicated servers represented by dashed-line boxes, but does not show the resulting complexity of interconnection. This complexity can be managed automatically by SNG. This application-decomposition approach is simple because the data are sent out to "specialist servers" for computation and then return, to be gathered and sent out to other specialist servers for other computation. More sophisticated approaches allow the data to flow through a network of specialist servers, being distributed and subsequently gathered back together along the way. This could eliminate the potential bottleneck of the central servers shown in Figure 14. Another approach is to duplicate the central server sufficiently, to balance the system.

## Software-development cost considerations

The development costs of large engineering software systems depend directly on the size of the development team and the duration of the implementation process. Obviously, if technology enables a team of software developers to perform a task faster, or if the technology enables a smaller team to perform the same task, the development cost is reduced. In addition, there are development costs that are incurred as a result of the

management requirements of the development team. Team size has a dramatic effect on management requirements. Large development teams require an elaborate management infrastructure to ensure project convergence. Some CASE tools address both the software development issues and the problems of managing development teams. The two CASE tools presented in this paper reduce the development task for significant applications so that fewer than three software developers and very little management structure are required.

## Conclusion

The brevity of APL2 as a vehicle for mathematical expression enables the engineering software developer to concentrate on the mathematical solution to a problem. This brevity and the resulting productivity enable the engineer to justify brief periods of software development, thus eliminating the need for programmer intermediaries.

The integration of powerful data management capabilities, such as generalized nested arrays, and an extensive range of mathematical tools in APL2, facilitated the implementation of the seamless design-to-manufacture system described. The capability of APL2 to treat data as executable software is central to the ESG approach of using automatically generated drive data to actually "drive" the expert system.

The ESG environment provides the developer with the brevity of graphical expression, to decompose an expert system in the context of the engineering application. Here too, the brevity and resulting productivity enable the engineer to undertake the implementation of a customized application system. The engineer avoids the effort and pitfalls of attempting to characterize identified needs, to formulate a system specification, and to depend on a programmer intermediary who is not expert in the application domain.

The effectiveness of the ESG system approach to the implementation of customized engineering application systems is due to the following factors:

1. The high level of automation achieved in the top-down specification, implementation, and maintenance mode
2. The high level of productivity achieved in the bottom-up mode due to the direct translation of mathematical solutions to APL2 expressions
3. The flexibility to switch back and forth between the top-down and bottom-up implementation modes
4. The availability, throughout the development process, of a working system prototype ready to be tested according to the current level of specification

The combined SNG, ESG, and APL2 environment extends the same advantages of brevity and graphical expression to developers of systems involving applications that can benefit from the increased computational speed of distributed cooperative processing. The need for computerization of the manufacturing floor, combined with the emergence of various powerful computer-hardware and communication platforms that are suitable for this task, make this an ideal application for the software-development environment. However, the applicability of the new hardware and communication platforms is not limited to manufacturing applications.

Both the geometric modeler and the automatic process generator described in this paper are structured to benefit from distributed cooperative processing. This is also true of many other engineering problem solutions. Conceptually, this approach applies to any engineering problem that may be decomposed in the form of a block diagram. Consequently, the combined SNG, ESG, and APL2 environment should be viewed as a generic software-development environment, which can be applied to significantly reduce the application-development and maintenance cost of engineering applications.

## Acknowledgment

All of the manufacturing process information, upon which this research is based, has been provided by C. Stanley, Senior Computer Integrated Manufacturing (CIM) Specialist at Boston University's Computer Integrated Design, Analysis, and Manufacture (CIDAM) Laboratory, based upon his many years of firsthand experience with high-precision machining throughout the aerospace industry. This research was funded during the past few years by Karsten Manufacturing Corporation, IBM Corporation, and by the General Electric Corporation. The CIDAM Laboratory's research testbed of IBM 7437 Virtual Machine/System Product Technical Workstations was provided

by IBM through a joint research contract (MHVKUO51).

*Trademark or registered trademark of International Business Machines Corporation.

## Cited references


1. Y. Hazony, "Design for Manufacture: A Geometric Modeler for Geometric Integrity," *Manufacturing Review*, American Society of Mechanical Engineers **4**, No. 1, 33–43 (1991).
2. L. Zeidner, "Automatic Process Generation and the SURROUND Problem: Solution and Applications," *Manufacturing Review*, American Society of Mechanical Engineers **4**, No. 1, 53–60 (1991).
3. Y. Hazony, "Application of Nested Arrays to Databases for Engineering Design," *APL91 Conference Proceedings, APL Quote Quad* **21**, No. 4, 197–201, ACM, New York (August 1991).
4. L. Zeidner, "Automatic Process Generation: The SURROUND Problem," *Proceedings of the 2nd International Conference on Computer Integrated Manufacturing*, IEEE Computer Society Press, Troy, NY (May 1990), pp. 331–338.
5. Y. Hazony, "Geometric Modeling for Design for Manufacture," *Proceedings of the 2nd International Conference on Computer Integrated Manufacturing*, IEEE Computer Society Press, Troy, NY (May 1990), pp. 131–136.
6. L. Zeidner, Y. Hazony, and A. C. Williams, "An Expert System Generator," *IASTED Journal of Control and Computers*, International Association of Science and Technology for Development, **15**, No. 1, 22–33 (1987).
7. L. Zeidner, Y. Hazony, and A. C. Williams, "Expert System Generation via the Data-Driven Control-Flow Methodology," *Proceedings of APL'87 Tutorials*, ACM, Dallas (May 1987), pp. 1–11.
8. *APL2 Programming: System Services Reference*, SH20-9218, IBM Corporation (1987); available through IBM branch offices.
9. L. Zeidner, "The Server Network Generator (SNG): A CASE Tool for Distributed Cooperative Processing," *APL91 Conference Proceedings, APL Quote Quad* **21**, No. 4, 369–385, ACM, New York (August 1991).
10. L. Zeidner, "Server Networks: A CIM Architecture Design Environment," *Proceedings of CIMCON'90 International Conference on CIM Architecture*, National Institute for Standards Technology, Gaithersburg, MD (May 1990), pp. 95–113.
11. L. Zeidner, "Server Networks: Distributed Simulation and Modelling," *Proceedings of IASTED Applied Simulation and Modelling '89*, ACTA Press, Anaheim, CA (November 1989), pp. 138–142.
12. L. Zeidner, "Server Networks: Software Integration Tools for CIM," *Proceedings of the International Conference on Computer Integrated Manufacturing (CIMIC)*, IEEE Computer Society Press, Rensselaer Polytechnic Institute, Troy, NY (May 1988), pp. 226–235.
13. L. Zeidner, "Server Networks Communicating Via Inter-User Shared Variables," *APL87 Conference Proceedings, APL Quote Quad* **17**, No. 4, 173–181, ACM, New York (May 1987).
14. R. H. Lathwell, "System Formulation and APL Shared Variables," *IBM Journal of Research and Development* **17**, No. 4, 353–359 (1973).
15. A. D. Falkoff, "Some Implications of Shared Variables," Technical Report TR-02.688, General Products Division, San Jose, CA 95193 (1975).
16. *APL2 Programming: Processor Interface Reference*, SH20-9234, IBM Corporation (1987); available through IBM branch offices.
17. *Virtual Machine/Pass-Through Facility, Managing and Using Release 4*, SC24-5374, IBM Corporation (1988); available through IBM branch offices.
18. *7437 VM/SP Technical Workstation: Programmer User's Guide and Reference*, SA23-0351, IBM Corporation (1988); available through IBM branch offices.
19. R. J. Sahulka, E. C. Plachy, L. J. Scarborough, R. G. Scarborough, and S. W. White, "FORTRAN for Clusters of IBM ES/3090 Multiprocessors," *IBM Systems Journal* **30**, No. 3, 296–311 (1991).


*Accepted for publication September 23, 1991.*

**Yehonathan Hazony** *College of Engineering, Boston University, 44 Cummington Street, Boston, Massachusetts 02215.* Dr. Hazony is a professor of manufacturing engineering at Boston University. He is the principal investigator on a Joint Research Agreement with the IBM Network Computing Group at Poughkeepsie, New York, concerning the application of 7437 VM/SP Technical Workstations to engineering. Dr. Hazony joined Boston University in 1984 to found and direct a laboratory for computer-integrated design, analysis and manufacture (CIDAM). His current research interests are in the conceptualization and implementation of seamless design-to-manufacture (SDTM), as a post-CAD/CAM technology. His research interests also include high-productivity software-development methods for customized SDTM systems. Prior to joining Boston University, Dr. Hazony founded the Interactive Computer Graphics Laboratory (ICGL) at Princeton University (1974) and directed it until 1983. This laboratory was developed, in collaboration with IBM, to explore the use of mainframe-based engineering workstations as a vehicle for the introduction of numerically intensive computation to engineering education and research. Dr. Hazony was also an associate director of the Princeton University Computer Center. Dr. Hazony received his Ph.D. in physics in 1965 from Hebrew University, Jerusalem, Israel.

**Lawrence Zeidner** *College of Engineering, Boston University, 44 Cummington Street, Boston, Massachusetts 02215.* Dr. Zeidner is an assistant professor of manufacturing engineering at Boston University. He has conducted research in the area of software-development methods for distributed cooperative processing (DCP), as part of a Joint Research Agreement with the IBM Network Computing Group in Poughkeepsie, New York. Dr. Zeidner's research interests are in the area of high-productivity software-development methods such as graphical programming, data-driven control flow (DDCF), and automatic code generation. His research also includes automatic process generation in the context of seamless design-to-manufacture (SDTM). He was the recipient of a General Electric Foundation Young Faculty Award from 1986 to 1989. Dr. Zeidner received his B.S.E. in 1980 in electrical engineering and computer science, his M.A. in 1982, and his Ph.D. in 1983 in civil engineering from Princeton University.

Reprint Order No. G321-5464.