HEWLETT-PACKARD
JOURNAL

MAY 1987

HEWLETT-PACKARD
JOURNAL

May 1987 Volume 38 • Number 5

Articles

Research Report

Departments

In this Issue

HP DesignCenter is the name given to a collection of design automation software packages that run on the HP 9000 family of technical computers. These products are tools that help electronic, mechanical, and software engineers be more effective in their jobs. What they do is often spoken of in acronyms—CAE (computer-aided engineering), CAD (computer-aided design), CAM (computer-aided manufacturing). On pages 4 to 33 of this issue, designers of the HP DesignCenter Mechanical Engineering Series 5 and Series 10 systems, our cover subjects, describe the structure and operation of these advanced two-dimensional design and drafting packages. Automating the process of designing mechanical parts presented numerous engineering challenges that required contributions in user interface design, memory management and data structure design, and algorithm design. For ME Series 10, an additional design challenge was the requirement to communicate with systems for finite element structural analysis and systems that translate part data into instructions for the numerically controlled (NC) machine tools that make the parts. The paper on page 4 describes the ME Series 5/10 products and their user interface. The system is designed to be both easy to learn for the novice and powerful enough not to limit or frustrate the expert. The command language and its user-definable macro facility are noteworthy. Data structures and algorithms for geometry construction, dimensioning, hatching, and parts definition are discussed in the paper on page 16. On page 30 is an unusual (for us) report from an HP Division that acted as an alpha test site for the ME Series 5/10 and is now using these systems to design printer parts.

Because it's already there, electric wiring has long been used for many things besides carrying electric power. In homes, for example, some cordless telephones, intercoms, music systems, appliance controllers, and burglar alarms use the power lines, saving the expense of separate wiring. Not surprisingly, people are now connecting computers over local area networks (LANs) that use the power lines for data communications. On page 35, we have a report on an HP Laboratories research project on power-line LANs. The research revealed significant obstacles to reliable data communications over power lines, including high noise levels and high signal attenuation that vary with time and from site to site. To deal with these conditions, a wideband scheme was tested and found to permit usable data rates up to 100,000 bits per second, much higher than previously reported.

-R. P. Dolan

What's Ahead

The June issue will cover a variety of topics, including HP's Human Interface Link (HP-HIL) for personal computer and workstation input devices, the design of a low-cost HP-HIL graphics tablet, interprocess communication mechanisms for UNIX systems, and software verification using branch analysis. Completing the issue is a Viewpoint article on the direction of CMOS technology by Yoshio Nishi.

The HP Journal encourages technical discussion of the topics presented in recent articles and will publish letters expected to be of interest to our readers. Letters must be brief and are subject to editing. Letters should be addressed to: Editor, Hewlett-Packard Journal, 3200 Hillview Avenue, Palo Alto, CA 94304, U.S.A.

State-of-the-Art CAD Workstations for Mechanical Design

Part of HP's DesignCenter, the ME Series 5/10 workstations simplify the creation of part drawings and the design of mechanical assemblies. A shared data base improves communication among designers on a project and the results can be formatted automatically for use by NC manufacturing machinery.

by Wolfgang Kurz, Dieter Sommer, Karl-Heinz Werner, Dieter Deyke, and Heinz P. Arndt

BUILDING A HIGH-PERFORMANCE CAD SYSTEM takes more than putting a number of wonderful features together. The majority of today's CAD (computer-aided design) systems for mechanical engineering can more or less do the job, but many of the systems demand an extensive amount of training before the user can become productive. Another concern is that as a user's proficiency grows, requests for modification, customization, and new features arise.

To resolve some of these difficulties and to provide designers with tools to aid them in their designs, HP has developed a family of workstations, the HP DesignCenter, for use in various technical fields. Two examples are the Mechanical Engineering Series 5 and Series 10 (Fig. 1) discussed here. These workstations represent one of a group of HP projects to develop software for various CAD systems and more general interactive graphic systems. Software reusability is a high priority in these projects. This not only saves substantial development effort, but also benefits the user by providing a uniform set (or subset) of functions across a range of products.

Another HP DesignCenter goal was building a CAD user interface that enables productive use of a system after only a few days, even for a novice without formal training. At the same time, the system does not disappoint an expert user. The ME Series 5/10 provides over 250 commands and functions with different options for many of them. A very efficient method of combining these primitives to form

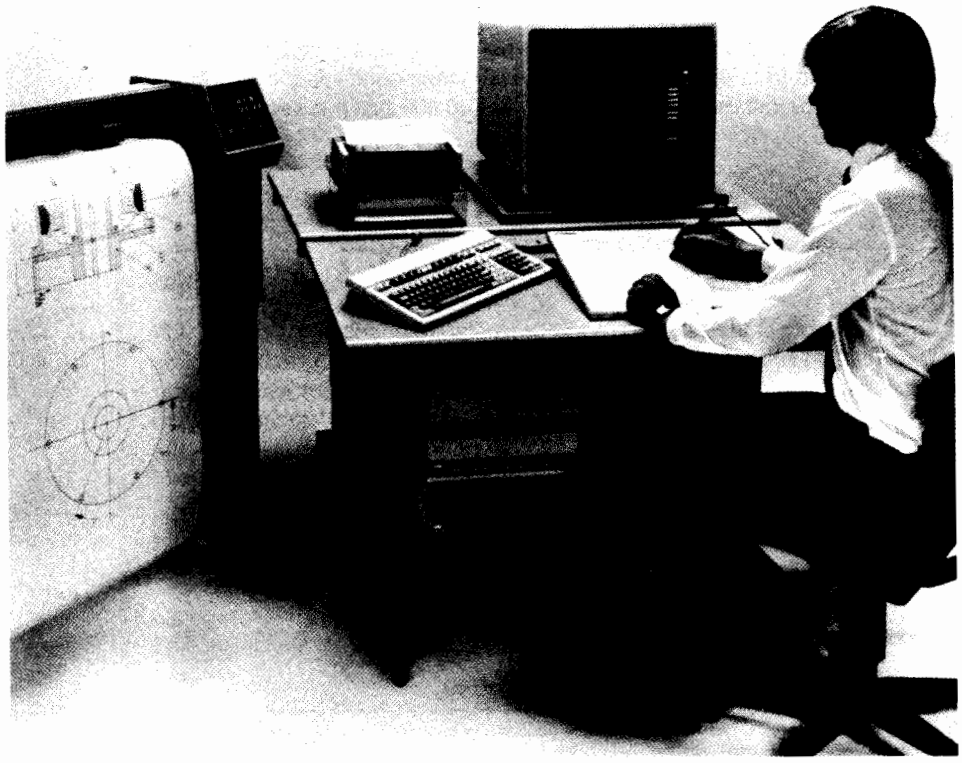

Fig. 1. *The Mechanical Engineering Series 5 and Series 10 Workstations of HP's DesignCenter are versatile systems for two-dimensional drafting (Series 5 and Series 10) and mechanical design (Series 10) applications. Both feature a friendly and easy-to-learn user interface designed for technical illustrators, drafters, and engineers. The Series 5 provides many tools for the preparation of drawings, flowcharts, schematics, layouts, and documentation and can be easily upgraded to the Series 10, which provides many facilities for mechanical engineering design such as parametric design evaluations, customizability, and interfaces to other CAD systems (e.g., for finite-element analysis) and to numerically controlled (NC) machinery.*

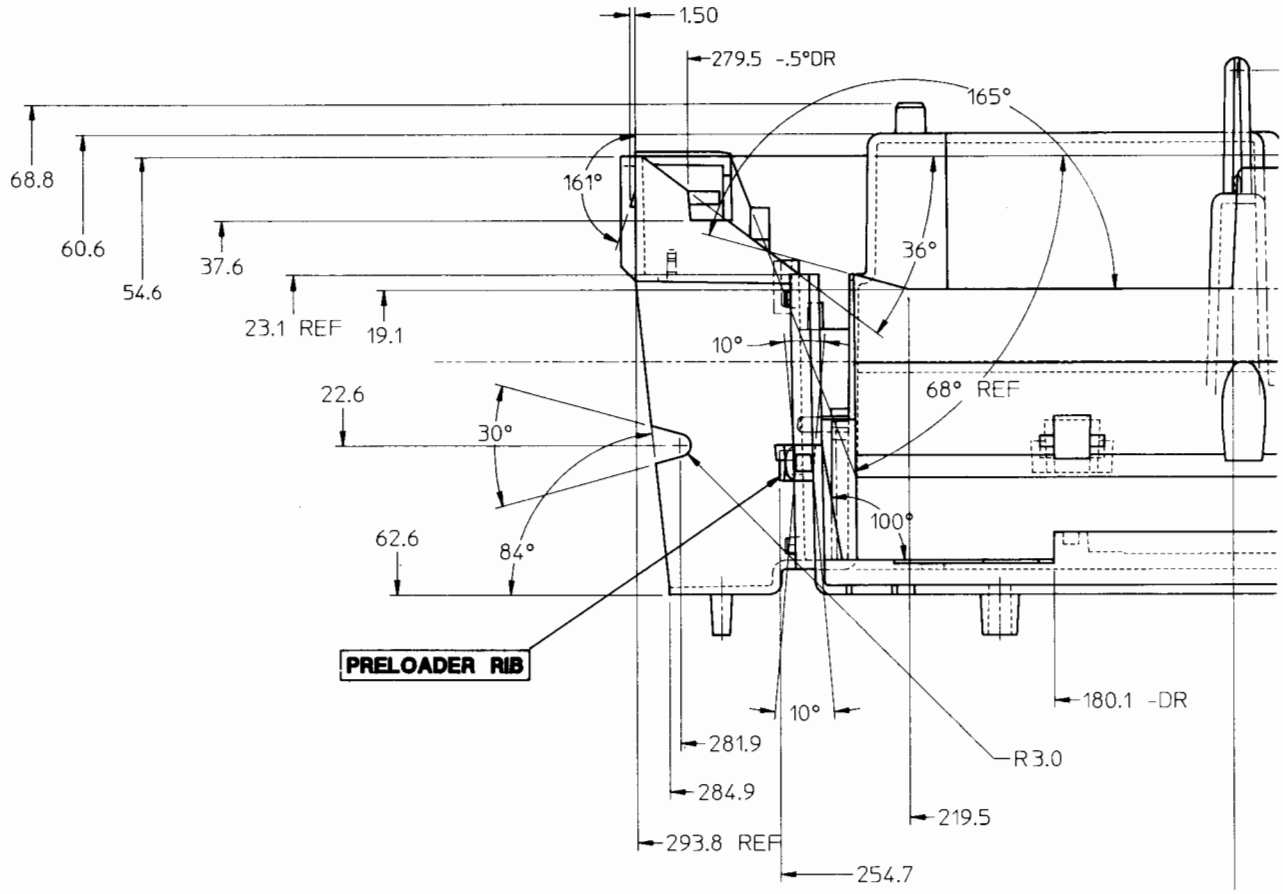

Fig. 2. *A typical ME Series 5/10 drawing.*

more complex command structures provides the experienced user with a powerful tool for daily design tasks.

Feature Highlights

The HP DesignCenter Mechanical Engineering Series 5 and 10 are advanced two-dimensional design and drafting systems for mechanical engineering applications. Fig. 2 shows a typical ME Series 5/10 mechanical drawing.

The ME Series 5 and ME Series 10 have the same software architecture, basic user interface, and model data structures. The ME Series 5 offers a subset of the ME Series 10 capabilities at an economical price. It is upward compatible with and upgradable to the ME Series 10.

ME Series 10 features include:

- Comprehensive construction and annotation capability
- Choice of user interfaces—tablet/screen or screen only
- Powerful command set to perform fast design modifications
- Sophisticated macro language for parametric design and design evaluations
- Customizability for specific applications
- Interfaces to other CAD systems (IGES), NC machines, and finite element analysis systems.

The ME Series 5 offers the same features with the following limitations:

- Limited drawing size
- Limited macro capabilities
- No editing for hatch parameters, attributes, or fillets
- No interfaces to NC, FE, or IGES systems.

ME Series 5/10 construction geometry simplifies the work previously done using a pencil, compass, and ruler on a drafting board. Construction geometry consists of basic figures like circles and endless lines. There is a mode to overdraw the construction geometry, like inking in the pencil lines. The user can also create real (inked) geometry directly. The set of geometry elements includes third-order splines (with damping) which can be converted into a contour of tangent lines and arcs if the geometry is to be transferred to an NC system (numerically controlled automatic machining equipment). Besides the easy handling of texts and symbols, other annotations such as hatching and dimensioning not only comply with different standards like ANSI, ISO, DIN, or JIS, but are sophisticated enough to be updated automatically when the associated geometry is altered. The ability to create assemblies and hierarchically structured parts simplifies the handling of large amounts of data. Since the data base can be shared with other designers at other ME Series 5/10 workstations, each designer can always have ready access to other parts affecting the designer's particular design. (See the article on page 16 for details about ME Series 5/10 geometry, dimensioning, hatching, and parts.)

The ME Series 5/10 products are tailored for the HP 9000 Series 300 Computers,[1] but are also supported on the HP 9000 Series 200 Computers. They take full advantage of the different hardware configurations and the broad range

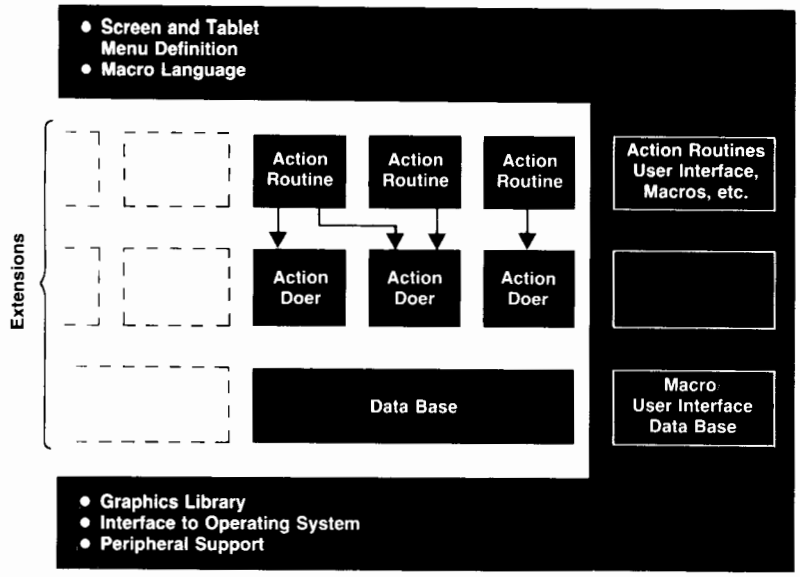

Fig. 3. *Software architecture of the ME Series 5/10.*

of supported peripherals. The operating systems the products run under are the HP-UX operating system and the Pascal workstation systems (Series 200). An interface to each operating system is provided to take advantage of all of the features of the operating system.

The hardware configuration can be as simple as a single stand-alone 16-bit workstation using a medium-resolution monochrome display and a mouse as the input device. For more sophisticated applications, it can be extended to a network of 32-bit, floating-point workstations with high-resolution color displays, B/A3-size graphics tablets, printers, a choice of disc drives with storage capacities from 270 kbytes to 571 Mbytes, and E/A0-size plotters with automatic paper feed to provide multiple frame plots.

Customers with special applications need to customize their systems. The ability to run other programs on-line, together with a macro programming capability, makes the ME Series 5/10 useful to customers other than those in the mechanical and electromechanical design fields. Various macros, symbols, text fonts, and standard parts are available.

Customizing also means being able to use the system in the user's native language. Localized versions for France, Italy, and Germany are available. A Japanese version with approximately 4000 Kanji characters is available; it supports 16-bit Kanji characters in the editor, macros, menus, help subsystem, and text commands.

ME Series 5/10 is designed to restrict the user as little as possible. For instance, the number of elements in the model or drawing is bound only by the memory (or virtual memory) available. The number of macros, screen tablet menu fields, simultaneous text fonts, or symbols is limited only by the underlying operating system.

The ME Series 5/10 is compatible with earlier HP CAD products like HP Draft, HP EGS, and HP Design. The valuable data created by these systems can be transferred to the ME Series 5/10. An open system architecture not only means having interfaces to other HP products, but also an IGES (Initial Graphics Exchange Specification) interface to systems made by other manufacturers.

Software Architecture

The design goal for the software architecture was to develop a uniform user interface and operating system interface for all commands and functions, and mechanisms for additional commands, new applications, and different data bases. The U-shaped area in Fig. 3 contains all the software dealing with the user interface, customizing, command parsing, I/O device support, and general operating system support. Action routines define all user interaction for a specific command or function. The command's syntax is only defined inside the action routine. Action doers are more general modules containing mathematical algorithms and various calculations.

User Interface

The large functionality built into ME Series 5/10 is accessed by means of a command language with a defined syntax. However, for more efficient work and easier learning it was necessary to create a friendly environment for all commands and functions. There are combined graphics tablet/screen and screen-only user interfaces. All tablet and screen menus have been defined with ME Series 5/10 commands using the ME Series 5/10 macro language and can be easily customized by the user.

Catching a Point. The ME Series 5/10 catch mechanism allows the user to catch (select) a point in various ways. The point caught is the closest point of a preselected type within the circular area around the cursor center. It may be the closest vertex (end point), intersection, grid, or perpendicular projection point of the cursor center onto an element, or nothing. The catch mode can be changed at any time. Hence, while creating a polygon, it is possible to change the catch mode from vertex to intersection to get the third polygon point correctly. The window can similarly be changed to get a point outside of the current window. Catching is confined to the cursor area boundaries. These boundaries can be set by the user, who can always see the cursor on the screen and has complete control of the catch results. Hence, it's not possible to catch an intersection point in the upper right corner of the viewport by

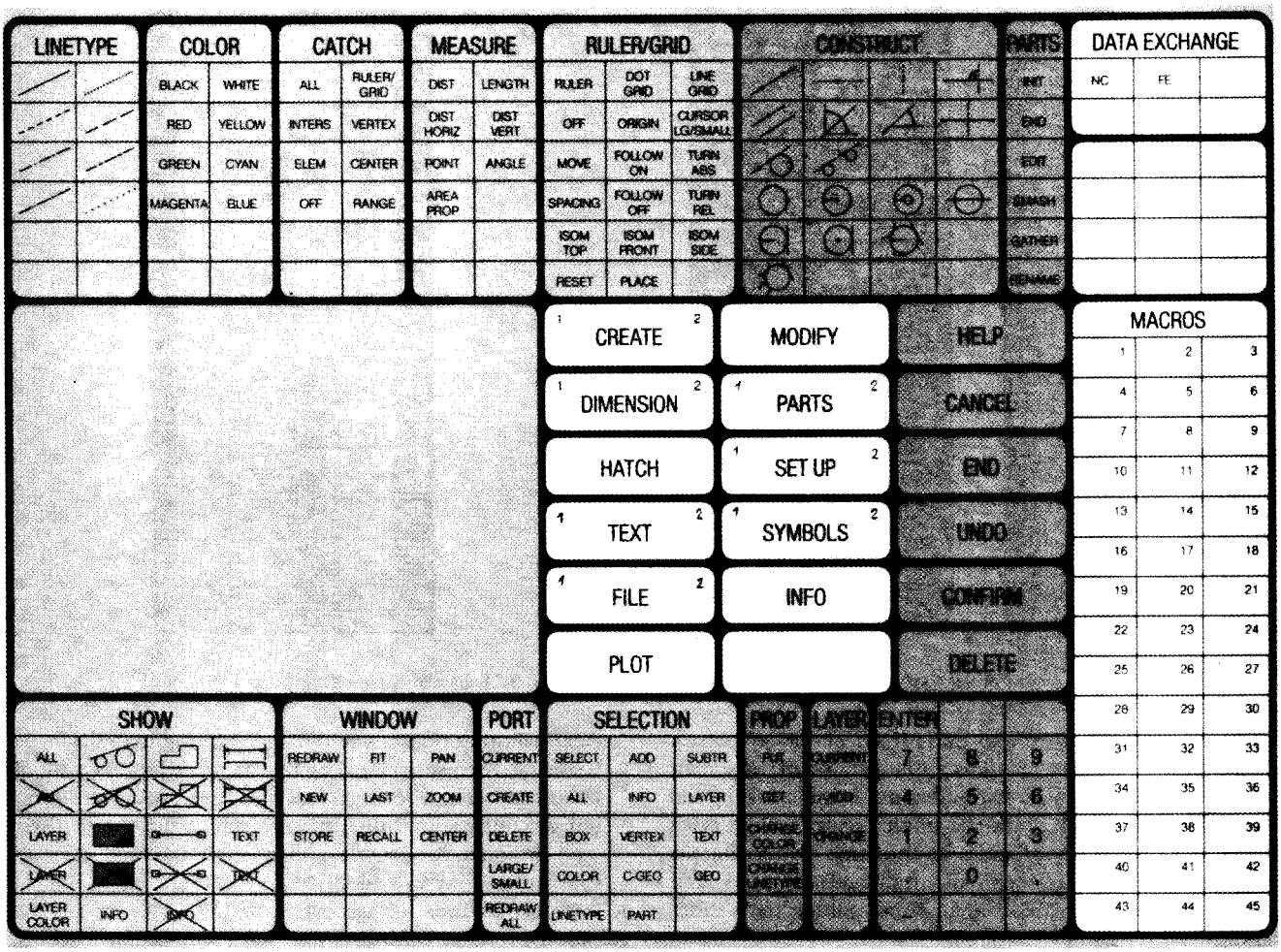

Fig. 4. *ME Series 5/10 tablet overlay.*

indicating a point in its lower left corner when the cursor size is five pixels.

Tablet/Screen User Interface. The ME Series 5/10 tablet area can be used for two different purposes: to pick the geometry points and to choose commands or functions. An overlay (Fig. 4) indicates the softkey locations on the tablet.

Following the desktop metaphor, we put all commands and functions that should always be available on the graphics tablet, while the items that are mainly used in certain design phases are found in screen submenus. This defines a clear and easy-to-learn structure for the user interface and makes its use much more effective. For example, a user can pick window functions or construction geometry commands from the tablet at any time without needing to change the CREATE screen submenu for that purpose.

The tablet overlay consists of:

- A column with often-used commands: DELETE, CONFIRM, UNDO, END, CANCEL, and HELP.
- Functions that can interrupt the active command at any time and functions for display options. Examples are SHOW, WINDOW, CATCH, RULER/GRID, and COLOR.
- Selection items that are normally used together with certain commands.
- A numeric keypad.
- Two columns with screen submenu calls.

On the tablet overlay, commands have a dark gray background color, functions and selection items are placed on a light gray background, and screen submenu calls are printed on a white background.

All commands and functions that are mainly used in special design phases have been collected in screen submenus with one or two screens each. The screen submenus are:

- CREATE (two screens). General drawing commands for real geometry. Hence, a user can create real geometry and construction geometry at the same time since the construction geometry block is placed on the tablet overlay.
- DIMENSION (two screens). Automatic or manual dimensioning and unit conversion.
- HATCH. Automatic or manual hatch, and simple hatches and hatch patterns.
- TEXT (two screens). Editing of texts and leader lines and font selection.
- FILE (two screens). Loading, storing, directory catalogs with sorting items, and general file operations.
- PLOT. Local plot and spool plot.
- MODIFY. Moving, copying, rotating, mirroring, scaling, stretching, and isometric modification of real geometry.
- PARTS (two screens). Commands for a single-level or mul-

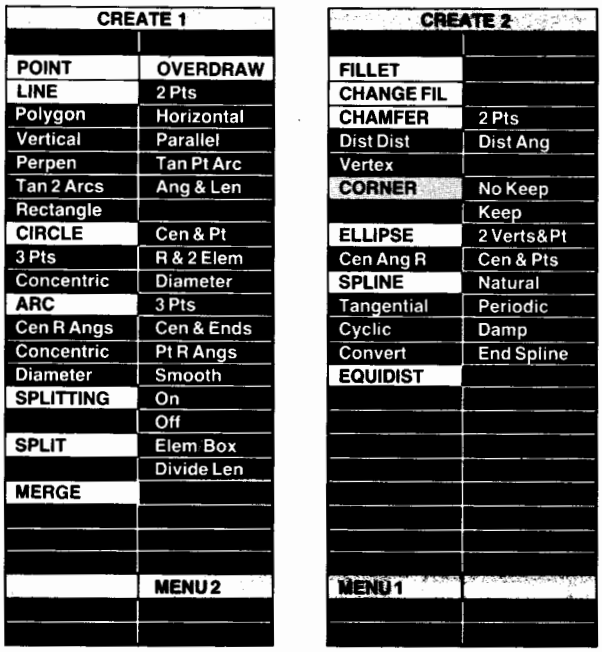

Fig. 5. *Screen submenus of the tablet/screen user interface.*

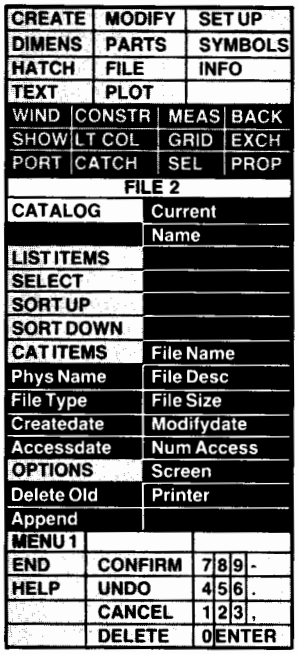

Fig. 7. *Example: Catalog function.*

tilevel parts structure and parts lists. The PARTS block on the tablet overlay contains a small subset of the screen submenu functionality and offers only single-level parts commands. This enables the novice user to work with a flat tree structure (all parts on the same level) whereas the experienced user can use the more sophisticated parts commands of the screen submenu to create more complicated parts structures.

- SET UP (two screens). Commands for changing the working environment, such as setting units, plotting defaults, dimensioning options, and dimensioning defaults.
- SYMBOLS (two screens). The first menu contains a set of ISO symbols in different sizes and the appropriate symbol box elements. Since many users want to use their own symbols, the second screen is left empty to offer an easy-to-use way to add custom symbols.
- INFO. Commands for associated texts.

Screen-Only User Interface. For users who prefer working with a mouse or do not want to spend money for a tablet, we implemented a screen-only user interface. Our main

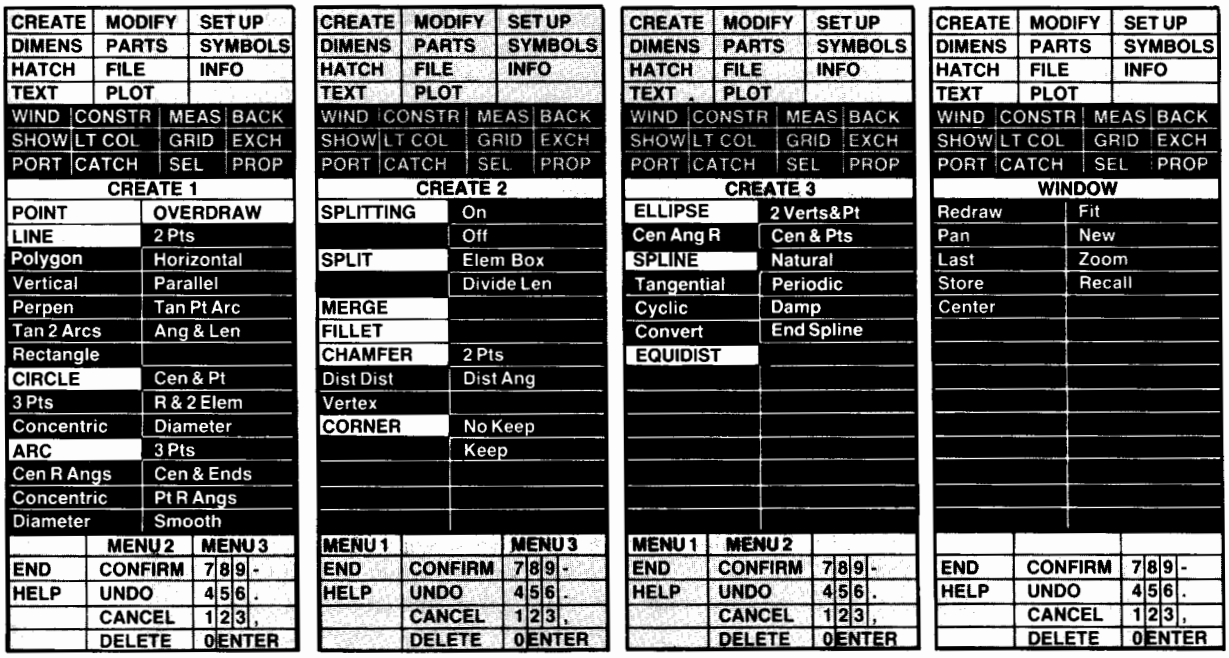

Fig. 6. *Screen submenus of the screen-only user interface.*

goal was to design both user interfaces to be as similar as possible by using the same structure for tablet overlay command blocks and their screen-only representation, and the same internal structure of the screen submenus like CREATE or DIMENSION. The scheme for the screen-only user interface consists of five parts:

- Calls for changing screen submenus. For example, CREATE and DIMENSION as you would find them on the tablet overlay.
- Calls for changing screen submenus that simulate the tablet overlay function blocks like WINDOW, SHOW, LINETYPE/COLOR, and SELECTION.
- An area in which the changing screen submenus will appear.
- A block with the main commands: DELETE, CANCEL, UNDO, CONFIRM, END, and HELP.
- A numeric keypad.

With the exception of the changing submenus, all other menu parts remain on the screen all of the time. Each menu part has a different background color. One additional function has been implemented in the screen-only interface. The BACK function leads the user directly to the previous command submenu after working in a function submenu like LINETYPE/COLOR.

Screen Submenus. Both user interfaces use several screen submenus (Fig. 5 and Fig. 6) with the same internal structure of commands. This makes documentation much easier since one manual describes both user interfaces.

Some of the menu items are enhanced by a background color selected according to a simple scheme. Normally, a command (white background) offers several options (black or blue background). If a user picks the command name itself, the user gets the default option that is always positioned to the right of the command name. To select an option, the user can pick it directly without picking the command name first. However, some commands or functions can be executed with more than one option at the same time. For example, let's take a closer look at the catalog function (see Fig. 7).

First, the user decides whether to see the current directory (pick CATALOG or Current) as the default option or another directory (pick Name and enter a directory name). Second, the user can select one or more options with a blue background to select certain catalog items (e.g., File Type), or sort the output list according to an item like file name, file size, create date, and so on. Finally, the user has to decide the destination for the catalog list. For this purpose, a blue background menu field named OPTIONS at the bottom of the screen submenu offers Screen and Printer as devices or Delete Old and Append for the destination file specified.

The different background colors in the screen submenus are used to simplify interpretation of the listings presented. A white background is used for command and function names. A black background shows necessary options that the user has to select with the first pick. Blue background options can be selected after the first pick.

User Interface Files. The user interface is completely written using the ME Series 5/10 macro language. It is divided into nine files which can be easily modified by the user:

hp_macro: All macros used in both user interfaces.

hp_macro_t: Macros and screen menus used only by the tablet/screen interface.

hp_macro_s: Macros and screen menus used by the screen-only interface.

hp_menu_t: Definition of the menu slot layout for the tablet/screen interface.

hp_menu_s: Definition of the menu slot layout for the screen-only interface.

hp_tmenu: The tablet overlay menus for three HP graphics tablets.

OVL9111, OVL46087, and OVL46088: ME Series 5/10 drawings of the tablet overlays for the three HP tablet types.

ME Series 5/10 Language

In addition to screen and tablet menus, special function keys are available. This highly customizable user interface system is based on an interactive command language chosen to be independent of input devices and input techniques and to provide an easily extensible system. Besides the basic commands and functions for invoking the system features, there are extensions for expression evaluation and macro definition.

Basic Constituents. Processing ME Series 5/10 input can be split into two parts: character processing and token processing. All logical input devices, such as the keyboard or a screen/tablet menu, send characters to the scanner. The scanner understands the syntax of all token types. It decides by looking at the stream of characters if the token is a point, a string, a number, et cetera. For example, 4,5 is a point, "ENGINEER" is a string, and 3.14 is a number.

A token is an entity of information (represented by a modified Pascal record) containing the token type (e.g., a number), its value (e.g., 3.14), and a link to the next token to be processed. The value of a token is in binary format, the value of a command token is an internal command number, and the value of a macro token is a pointer to the macro definition. This format was chosen because it is a compact representation and, since tokens are normally processed more than once, processing time is saved because most of the work is done when the token is built. The possible token types are listed in Table I.

Table I
Token Types and Examples

Token Type	Example
Command	LINE
Function	WINDOW
Qualifier	DOT_CENTER
Pseudocommand	LOCAL
Math function	INQ
Symbol	*
Macro	Center_lines
Literal	"Identify circle"
Number	3.14
Point	3.14,403
3D point	0,3.14,403
Illegal	1,a

Commands and functions are the primary keywords that

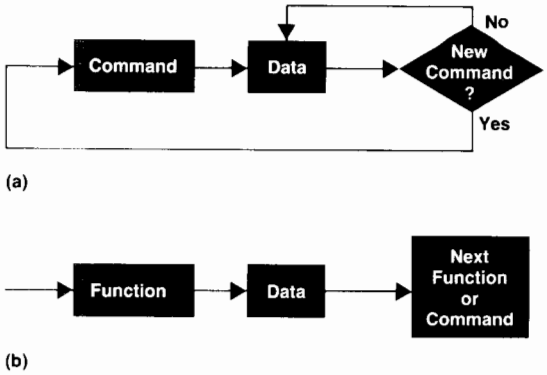

Fig. 8. *(a) Typical command syntax. (b) Function syntax differs from the command syntax in that it lacks the typical loop structure.*

invoke a specific system action. We have chosen different terms to reflect the two uses. A command (e.g., LINE) is normally active until another command is entered. This means that the user inputs lines until another command is chosen. The typical command syntax is shown in Fig. 8a.

A function (e.g., WINDOW) is normally only active until the data required for this specific function has been entered. Then the function terminates (see Fig. 8b). It does not have the typical loop syntax of the commands. Functions are those actions in the system that do not change the data model, but may change the values of variables defining the environment, such as a catch mode.

A special property of a function is that it can interrupt a command. That means that at any stage of the command, instead of entering the required data for that command, the user can select any function. After completing the function the system will ask for data for that command again. Even inside a function, another function can be invoked, which again interrupts the previous function.

Both commands and functions are implemented in action routines (see Fig. 3).

Expression Evaluation and Macro Processing. Between the scanner, which performs syntactic analysis and generates tokens, and the action routines, which consume tokens for semantic analysis to carry out the requested actions, is a filter to evaluate expressions and expand macros. Expression evaluation transforms several tokens into a single token. Macro expansion takes a single token and replaces it with one or more tokens. For example, the seven tokens $3.14 \times 10 \times 10$ are replaced by the token 314 and the macro token Dac is replaced by the tokens DELETE ALL CONFIRM.

Examples of expression operators are +, which adds two numbers or vectors or concatenates two strings, ROT, which rotates a vector, and POS, which returns the position of a substring within a string. One special operator is INQ. It can be used to read the current environment (e.g., window setting, color, catch mode, and available memory), the type and parameters of drawing elements (e.g., coordinates, radius, and color), and miscellaneous items (e.g., last measured distance).

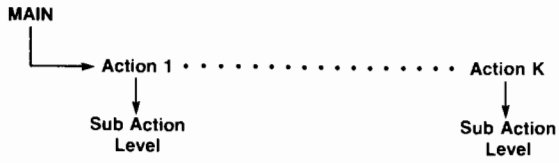

Fig. 9. *General program structure of ME Series 5/10.*

Macros can be used for the following reasons:

- Abbreviations such as Dac for DELETE ALL CONFIRM
- Speed (executing a macro is much faster than executing the same commands given in character representation)
- Localization. For example, using German KREIS for English CIRCLE
- Variables such as point coordinates, loop counters, etc.
- User-defined functions to extend the built-in functional capabilities
- User-defined action routines to extend the overall system capabilities, such as an ellipse routine
- Parameterized library parts such as bolts and nuts.

The macro features include nesting (macros can use other macros), local macros, and parameters. To avoid conflicts with a global macro with the same name, a macro should be declared local if it is only used inside another macro. Parameters are the same as local macros, except that an initial value must be provided when calling the macro. There are also several control structures to be used inside macros. These are WHILE END_WHILE, REPEAT UNTIL, LOOP EXIT_IF END_LOOP, and IF ELSE_IF ELSE END_IF. User interaction is provided by the READ function, which includes type checking, rubberbanding, catching, and default values. Macros can be not only saved in ASCII representation, but also stored in a binary format and secured as appropriate. The TRACE function can be used to monitor macro expansion and expression evaluation. It lists all tokens processed by the macro/expression filter either to an output device or to a file.

A short example in the box on page 12 explains the ME Series 5/10 macro capabilities in more detail.

Action Routines. Action routines are the lowest-level user interface in the system. They cannot be changed by the user, and within the same product and revision, they are the same for all countries.

The general program structure of ME Series 5/10 can be visualized by the diagram in Fig. 9. An action routine interfaces the user to the sub action level. For example, to create a line, the user enters LINE into the system (this can be done in various ways). LINE is recognized by the system as

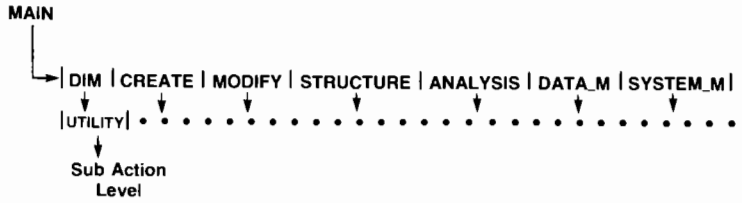

Fig. 10. *Expansion of action routine structure in Fig. 9.*

a command and the line action routine is called by MAIN. This routine now handles all interactions with the user to create the line the user wants. In general, an action routine reads a specific set of tokens and changes the state of the system accordingly. The state of the system includes the values of global variables, local variables, data structures, the file system, et cetera.

Fig. 10 expands the action routine structure and shows some of the ME Series 5/10 action routines. UTILITY denotes a set of action routines used by other action routines. Basically these routines deal with selection. Other action routines shown in Fig. 10 are:

- DIM, which handles dimensional data
- CREATE, which creates geometric information
- MODIFY, which deals with DELETE, SPLIT, MERGE, and MODIFY
- STRUCTURE, which deals with parts and layers
- ANALYSIS, which calculates properties of the created geometry such as center of gravity
- DATA_M, which manages data structures (LOAD, STORE, and MI)
- SYSTEM_M, which manages system parameters, macros, et cetera.

Store and Load

Storage Formats. ME Series 10 can store drawings, subparts, or library parts in two different formats: binary or MI (Model Interchange). ME Series 5 offers only the binary format. The binary file format corresponds to the internal representation of drawings, subparts, or library parts in ME Series 5/10. Very little conversion is necessary when storing to another binary format. This results in compact data storage and provides for fast storage and retrieval of ME Series 5/10 model data. The MI format is an HP data exchange standard similar to IGES that requires transformation of the internal data structure to an ASCII format for long-term storage and interfacing to external programs.

Some techniques used to store data are applicable to both formats. In particular, the same mechanism is used to store lists and object references. Before explaining the way ME Series 5/10 stores model data, it is necessary to describe some aspects of the internal memory management and the data structure of ME Series 5/10.

Memory Management and Model Data Structure. All objects that are created in a dynamic fashion or vary in size are stored in the heap storage area. To manage the heap area, ME Series 5/10 uses its own heap manager to obtain sufficient performance for creating and deleting dynamic objects. This helps achieve a satisfactory interactive re-

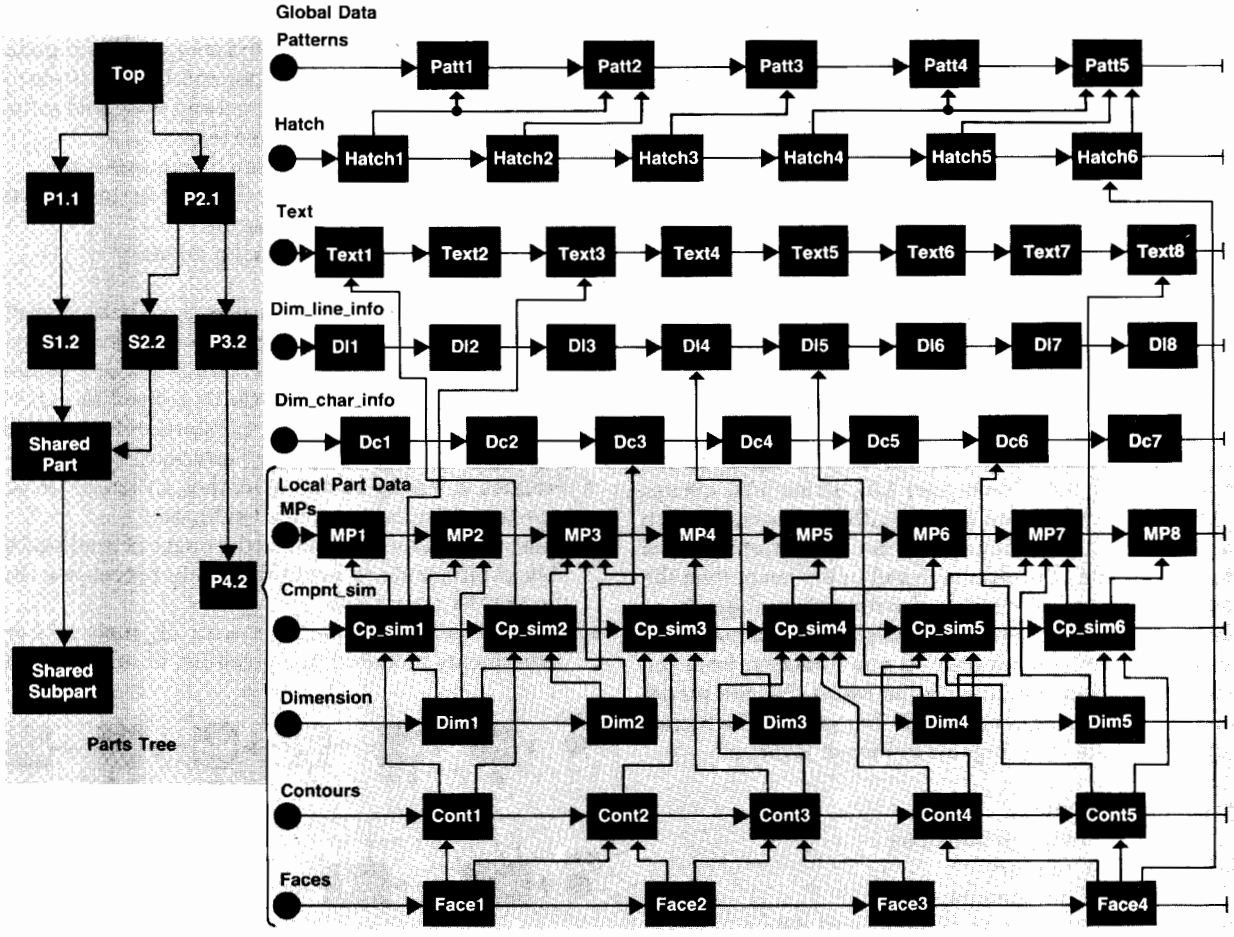

Fig. 11. *Objects are linked with references (black arrows) to one another.*

Example Macro

```
DEFINE Center_lines
  LOCAL P LOCAL R LOCAL C LOCAL L   (Declare macros used locally)
  END                               (Terminate any active command)
  INQ_ENV 3                         (Inquire current color and
  LET C (INQ 201)                    linetype, and save values
  LET L (INQ 302)                    in local macros)
  LOOP
    RGB_COLOR C                     (Restore color)
    LINETYPE L                      (Restore linetype)
    END
    LOOP
      READ PNT 'Identify circle' P  (Ask user to identify a circle)
      INQ_ELEM P                    (Get element parameters)
    EXIT_IF (INQ 403=CIRCLE)        (Exit loop if element=CIRCLE)
      BEEP                          (Else BEEP and continue asking)
    END_LOOP
    LET P (INQ 101)                 (Inquire circle center point)
    LET R (1.1*INQ 3)               (Inquire radius, scale by 1.1)
    LINE
      YELLOW                        (Switch color to YELLOW)
      DOT_CENTER                    (Switch linetype to DOT_CENTER)
      (P-PNT_XY R 0) (P+PNT_XY R 0) (Draw horizontal center line)
      (P-PNT_XY 0 R) (P+PNT_XY 0 R) (Draw vertical center line)
  END_LOOP                          (Ask for next circle)
END_DEFINE
```

The macro listed above will continuously ask the user to identify a circle to add center lines to. The center lines will be yellow, in a dot_center linetype, and will be 10% longer than the circle radius. The input of any command will terminate the macro and execute the command.

sponse time in ME Series 5/10.

The storing of objects organized in a heap area takes longer than storing an array of bytes containing all objects compacted with no free space between the objects. Such compact arrays of bytes can be copied from memory to disc in a very fast manner. But, in ME Series 5/10, all important data structures are organized as linear lists, binary trees, quad trees, part trees, etc., with many references between the objects. Using a heap storage area offers much more flexibility than reserving a fixed amount of storage organized as an array. The size of the model data is only limited by the installed memory.

To get the required functionality and to represent the dependencies of the objects that ME Series 5/10 creates and uses, most objects are linked with references to each other (see Fig. 11). If there were no restrictions on the references between the objects stored in the heap, then the net of objects could be described with a general graph.

ME Series 5/10 organizes the model data as a hierarchy of objects with references only in one direction from the higher to the lower levels, with the result that the model data can be seen as a directed cycle-free graph. The hierarchy of model data from highest to lowest levels is:

Highest level ⇝ Parts
Faces, dimensions
Contours
Components
Lowest level ⇝ Model points and global data

Because of this data organization, the storing of components and other objects can be simplified and done more efficiently.

All objects can be divided into two groups. There are objects global for all parts (shared by all parts) and objects local to a particular part, to which they belong. Examples are:

Global objects:
- Dimension line data
- Dimension character data
- Hatch
- Patterns
- Associated text

Local objects:
- Model points
- Components
- Dimensions.

Shared object references are used to link local objects to global object data and to link local objects in a part to each other as listed below:

Simple components	⇝ model points
All components	⇝ associated text
Contours	⇝ simple components
Faces	⇝ contours
	⇝ hatch
Hatch	⇝ patterns
Part	⇝ simple and composite components (all components belong to one part)
	⇝ part (shared and unshared subparts)
	⇝ associated text
Dimension	⇝ dimension line information
	⇝ dimension character information
	⇝ model points
	⇝ simple components

For many temporary processes, each sharable object has a slot called temp_attr (temporary attribute). The information stored in temp_attr is only valid within a single command or function. The temporary attribute is used to:

- Number objects in a list to transform references, mark objects, and assign temporary information.
- Produce a listing of the assembly structure or of the number of occurrences of all subparts.
- Temporarily store pointers to the image of modified sharable components to avoid duplicate processing during the MODIFY command.
- Find all faces and contours with all their referenced components included in a given list of selected compo-

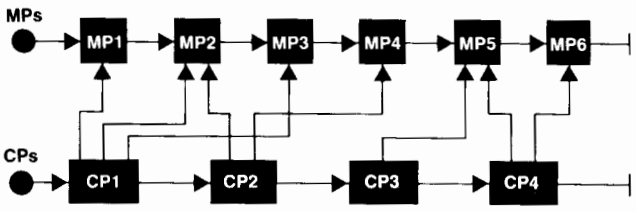

Fig. 12. *Lists of model points and components with references from the latter into the former.*

nents.

- Detect cycles during automatic hatching.
- Temporarily store topological information during the STRETCH command.
- Temporarily mark selected elements during selection and support logical operations in the selection mechanism.

Example: Storing Lists of Objects. As an example, we describe the mechanism to store two lists of objects with references from the second list into the first list. Let us assume that the first list contains model points (MPx) and the second list contains components (CPy) as shown in Fig. 12. To store this structure the following actions are performed:

1. Initialize the temp_attr of all objects in list MP to zero.
2. Mark all model points referenced from components in list CP.
3. Order all marked model points by assigning to temp_attr a sequence number according to the position in list MP and remember the total number of model points.
4. Store the total number of model points followed by the model points in list order. The model point count is used to detect the end of the list during load and to calculate the number of bytes needed to store object references into this list.
5. Count the number of components in list CP.
6. Store the count of components followed by the components in list order and convert model point references by storing the value of temp_attr accessed by dereferencing the model point reference.

Storing the Data Structure. Before any data is stored on external files, the model data is inspected to check that all references point to legal objects. This is accomplished by adding a sequence number to the temp_attr of the particular object and verifying that all referenced objects have a sequence number in the appropriate range (between zero and the number of elements of the particular object type).

All quad trees for model points are reorganized to delete model points that are not leafs and that are no longer referenced by other objects (i.e., all use counts are zero). The leaf nodes that have no other references have already been deleted.

This mechanism is used for all lists representing the model data. The order in which lists are stored is important to guarantee that during the reload of these lists, all elements of other associated lists are already loaded when references to these elements are encountered. Translating a reference means to follow the reference and get the number in temp_attr assigned during previous numbering. These numbers are always relative to the beginning of the list the referenced element belongs to.

To get a compact storage format, the numbers are stored as one, two, or four-byte integers when the number of elements in a list is less than 2^8, 2^{16}, or max_integer, respectively.

All objects that can be referenced only one time need no special actions because the contents of these elements can be stored without pointer information. All objects of the same type within a part are stored together to eliminate storing the type flag for each element. Instead, there is one type flag stored at the start of the list of objects of the same type.

The structure information of the quad tree is also stored with the model points. At load time, the tree can be built very quickly without time-consuming reconstruction by insertion of each model point in a new quad tree. To store the parts structure, there is a small difference in the mechanism. Each reference to an unstored part initiates the storing of this part and the fact that it was stored is marked in the temporary attribute of the part by setting temp_attr to the sequence number of the part. The next references to this part are stored by using the sequence number read from temp_attr for this part.

Loading Data. The loading of the model data lists is done from the lowest to the highest level in the objects hierarchy. A transformation table is constructed for each list of sharable objects in which the address for each member of the list is stored on the index that equals the sequence number in the list. To resolve shared object references, the address of the already created object is retrieved from the appropriate transformation table. To minimize the temporary storage requirements, the lifetime of the transformation table is reduced to the time required to load one part.

Help System

A CAD system should not only be simple enough to be operated by a user who works with the system only now and then, but also sophisticated enough to allow a very experienced user to perform very special operations. In both cases there is a need for occasional assistance, for

```
================================================================================
STORE command

-->(STORE)-->+------>+-+--->(ALL)------>+-->+-------------->+->|file name|--->
             |       | |                  |   |               |
             '-(MI)->' '-->|partname|-->'   '-->(DEL_OLD)-->'

STORE stores the drawing to the named file.   If the named file already exists,
you must use the DEL_OLD option.  STORE can produce either of two formats.

MI selects the MODEL INTERFACE STANDARD format.  This is a text format intended
for long-term archiving and for interfacing with other systems.

The default format is an internal binary format.   Files  in  this  format  are
smaller than MI files, and can be written and read faster.

Files in either format are readable with LOAD.

ALL  means  to  store the entire drawing, from the top part down.  You can also
name a part, and only that part will be stored.
================================================================================
```

Fig. 13. *Example from the* help *file.*

ME Series 10 Link to HP-FE

The finite element method allows the engineer to analyze the behavior of a design accurately before it is manufactured. The design process can be considered as a two-step loop. The first step is the creation or modification of a design. The second step is the analysis of the design. If the analysis gives negative results, both steps have to be repeated.

HP-FE is a general-purpose finite element system for linear structural and thermal analysis of two-dimensional, symmetric, and three-dimensional structures consisting of linear elastic material with homogeneous, isotropic material properties. Its analysis capabilities include linear elastic, linear dynamic, and linear heat transfer solutions.

The program performs in three steps: input of necessary data, calculation, and display of results. The input process involves two steps: describing the geometry of the structure and defining the finite element mesh. The user only needs to enter enough points to define the boundary of the structure. Given the boundary of the object, a cross partitioning into various subregions must be specified. The geometry data consisting of points and lines (polygons) for each subregion can either be defined directly in the preprocessor or transferred from the ME Series 10 system.

The ME Series 10 data exchange program extracts and reformats the geometrical data required for the HP-FE program. In the ME Series 10 environment, the user chooses the drawing. For manipulations such as partitioning, the whole ME Series 10 modification capabilities can be used. Then the geometry can be passed in a well-defined manner to the preprocessor of HP-FE. The data exchange sets the third coordinate equal to zero, thus embedding the two-dimensional ME Series 10 data in an X-Y plane through the origin. Because of different operating systems, the file format is ASCII. After the transfer the user switches to the HP-FE program where it is possible to add finite element specific information. This consists of:

- Definition of the mesh
- Properties of elements
- Boundary constraints
- Definition of loads.

Entering this data permits finite element analysis to be performed.

Guenter Voss
Development Engineer
Böblingen Engineering Operation

example, to give the former user a hint about the order of input and to show the latter user a different way to deal with problems most efficiently. The assistance should give guidance, such as when a user just forgets how to input the correct command sequence, and should give hints about other commands or options that could be used as a better solution for the user's problem.

A configurable and expandable system that can be modified by the user should also allow the user to incorporate information about a specific functionality into the help system. The information presented should be technically correct and comprehensive. The technical user should find a notation that allows the user to check the syntax of commands, functions, and macros at a glance. Therefore, we chose the so-called railroad notation, where the user can travel along and find all options of a given command and the sequence in which they have to be entered.

The ME Series 10 help system is made up of three files. One, called help, is shipped with the system. The other two files are generated by the system and are called help.i and help.t. One is an index file that allows very fast access to the formatted text file—its contents are actually displayed on the screen.

All the information about the help system is in the master file called help. This file is human readable and can be edited with a regular text editor. To explain in more detail, let's take a closer look at a description of a specific command. Fig. 13 is an original example from the help file. The keywords are preceded by the caret ^. The keywords are later stored in the index file and used as keys to the corresponding help paragraph.

The first keyword is the primary keyword, which normally is the same as the command name. The other keywords may be applicable to more than one command. To get from paragraph to paragraph containing these keywords, the help command N (Next) is provided.

The railroad structure shows all the options and the command sequence. Words in capital letters (e.g., ALL) are known words of the system and have to be entered literally. The railroad structure is followed by a more verbal description of what the command and the options do, hints about performance, boundary conditions, and what happens if an error occurs.

The user can write paragraphs and insert them into the help file. The system then generates a new index and text file so that the user's paragraph is available from then on. Searching through the index file is very fast since the index file is loaded in main memory. If an exact match occurs, the key is found, and the paragraph is displayed. If no exact match is found, a second pass through the index file is initiated to find the best match. This means that if a simple typing error occurs or the exact syntax is not known, there is a good chance of getting the proper help message anyway.

Once a user has a help paragraph on screen, besides the above-mentioned Next command, the user can scroll the screen to read previous or following paragraphs. If the user is already inside a command or function and doesn't know how to proceed, the user simply can enter HELP, which automatically displays the help paragraph for the current command or function. The help system is always available wherever the user is in ME Series 5/10, whether in command entry, inside a command, in the editor, or even in the help system itself.

Development Process

The development took place on the Pascal workstation system using an HP SRM (Shared Resource Management) network, which is also a product configuration. 160,000 lines of Pascal source code, a few assembler modules, and some C language routines were completed in a relatively short time. A number of tools were written to automate and speed up day-to-day work. Continuous porting of the software to the HP-UX operating system and continuous testing during the development phase reduced the chance of unwelcome surprises at the end. Early prototypes and the feedback from our alpha and beta test sites (see article

The ME Series 10 NC Links

All NC (numeric control) programming languages are based on the same principle. First, geometric elements like points, lines, circles, and contours are defined. Then the tools are moved along these elements. For identification, all elements are labeled. For example:

```
P1 = POINT/15,20
L1 = LINE/15,10,30,80.2
C1 = CIRCLE/50,55.3,20
```

Traditionally, an NC programmer is given a set of engineering drawings of the part to be produced. The programmer has to understand and interpret the drawings accurately before proceeding to write the NC program. This is where much time is consumed and errors are made. Sometimes the amount of geometry definition in the NC source program is considerable and many typing errors are possible. Therefore, it is desirable to provide links from a CAD system to an NC programming system to minimize programming time and get code free of error. These links support NC programming by generating a list of geometric elements translated into the corresponding NC language as shown by the example above. In addition, the CAD user can also offer a drawing with the label information that corresponds with that used in the NC source code.

However, most NC interfaces to CAD systems will not support drawing changes. That means that any drawing change leads to totally different labeling of the elements. The result is that all machining commands in the NC source program must be rewritten. Not so with ME Series 10. In case of drawing changes all existing labels are not renamed, they are part of the model.

The following NC links are available:

- ANSI APT
- COMPACT II (Manufacturing Data Systems, Inc. or MDSI)
- NCGL (NC Graphics Language of MDSI).

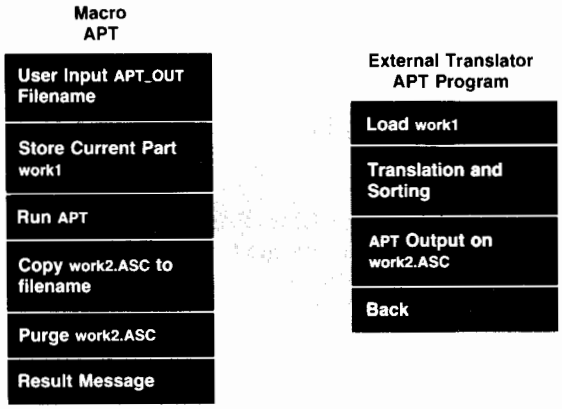

Fig. 1. *Translation process from ME Series 10 to NC language.*

Translation Process

The translation process is started by an ME Series 10 macro. The translators are implemented as external Pascal programs started by the RUN function of ME Series 10. This function suspends ME Series 10 and starts the external program (translator). Exiting the program will cause ME Series 10 to resume at the same point where it was left. The translation is based on MI data. If any labeled components exist in the drawing, only the labeled geometry is translated; otherwise, the entire geometry is translated. Fig. 1 shows the translation process.

Berthold Hug
Development Engineer
Böblingen Engineering Operation

on page 30) helped us tune the feature set and polish the user interface.

The extensive use of abstract data types made it possible to keep interfaces stable while experimenting with algorithms and data structures. For example, the point storage method changed from a linear list to a quad tree while in beta test. The test customers probably did not notice, except for the increase in speed.

Acknowledgments

Dan Matheson was the project leader of the breadboard that led to the ME Series 5/10. We thank Dan for his work. Mustafa Soliman was the first person outside R&D to work with ME Series 5/10 as a user, and he developed the first versions of the tablet/screen personality. Angela Suthurst of our publications group assisted with the editing and preparation of this article.

References

1. *Hewlett-Packard Journal*, September 1986, pp. 4-27.

ME CAD Geometry Construction, Dimensioning, Hatching, and Part Structuring

by Karl-Heinz Werner, Stephen Yie, Friedhelm M. Ottliczky, Harold B. Prince, and Heinz Diebel

CREATING MECHANICAL PART DRAWINGS is a necessary part of the design process for mechanical engineers and drafting and documentation support personnel. Using an HP ME Series 5/10 Workstation makes this task easier, simplifies the work required to make revisions later, and provides a data base that can be used by other designers and subsequent manufacturing facilities. The tools provided with the ME Series 5/10 allow a user to create and manipulate the following simple geometric entities:

- Construction lines and circles
- Lines
- Circles, arcs, and fillets (special arcs)
- Polygons and splines.

To represent these elements in the computer, the speed of geometric calculations and the amount of memory available are important constraints. To minimize the amount of storage required, we describe the elements above by vector algebra and express all quantities as multiples of real numbers:

- Point: 2 numbers
- Construction line: 3 numbers—for instance, one point on the construction line plus an angle of direction
- Construction circle: 3 numbers—for instance, center and a radius
- Lines: 4 numbers—for instance, two endpoints
- Arcs, fillets: 5 numbers—for instance, center, radius, and begin and end angles.

In the case of a construction line it is also possible to use the equation $y = ax + b$ to represent the (x,y) points of nonvertical lines.

Geometry Data Structure

The ME Series 5/10 data structure for geometry has a top-down structure with four levels:

- Level 3: faces
- Level 2: contours
- Level 1: construction lines and circles, lines, arcs and fillets, polygons, and splines
- Level 0: model points.

The Quad Tree. Repeated modifications like scaling a rectangle (consisting of four boundary lines) may introduce numerical errors into the values of the endpoints. It is then possible that the rectangle will no longer be closed. The best closure is obtained if the endpoints of adjacent elements are identical. A less stringent form of closure is to make the difference of the endpoints less than a given epsilon (ϵ).

To overcome the above-mentioned storage problem and the topological problem of closure, ME Series 5/10 uses the following mechanism. All points used to describe the geometric properties of elements are kept in a common storage area called model points. A model point is unique. There are points with topological relevance such as endpoints of lines and there are points with no topological meaning, for example, the center of a circle. To distinguish between these cases, use counters are attached to the points.

Entering a new point means comparing this point with other already existing points for identity. Two points are considered to be equal if both coordinate differences are relatively less than a variable called data_eps. The usual value of data_eps is about 0.000000000001 (10^{-12}). Comparing new points with existing points in this way could degrade the performance of the system, so an adequate storage format is necessary. The model points are stored in a quad tree.[1] The quad tree is a two-dimensional analog of a binary search tree. The basic idea of the quad tree concept is to associate four quadrants with a given point. Another point can then be classified to be in one of the four quadrants. Using the standard Cartesian coordinate quadrants, it is easy to classify points (see Fig. 1).

In the kth level of a quad tree there is room to address 4^k points. So a full quad tree of seven levels can store 21,844 points. The search time for a specific point is very short if the tree has good balance. Typically the quad tree

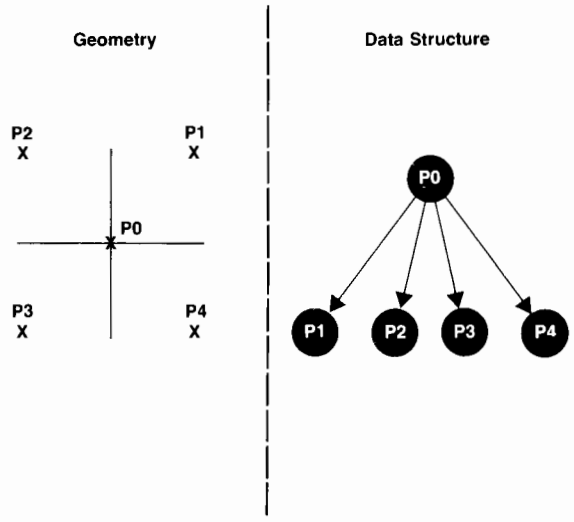

Fig. 1. *Quad tree concept.*

of a mechanical engineering object is balanced very well. If the model shape is equal to a nearly horizontally or vertically extended object, the quad tree degenerates to a binary tree. To degenerate the quad tree to a linear list, all points must be placed along a single line (a shape that is uncommon in mechanical drawings). The shape of the quad tree depends on input order. If an element referencing a given model point is deleted, then the use counters of the model point are decremented. If the use counters of this model point are zero and the program leaves the model point, the model point is deleted immediately. Otherwise, the model point is removed within the STORE command.

An inspection of some complex mechanical engineering drawings (about 5M bytes of data) showed the average tree level to be 20, with a few branches up to 50 levels deep.

With ME Series 5/10 a user can create points in the whole plane (limited by the longreal format) and the points may be arbitrarily close (again limited by the longreal format and by epsilon).

Level 1 Elements. A construction line is considered as an infinite line passing through a given point. The internal representation consists of a point and a direction vector. Construction circles consist of a center (point) and a radius. Lines are represented by two endpoints. Arcs and fillets are represented by a center and two endpoints. Circles are considered as arcs with identical beginning and ending points; hence, they are represented by a center and one peripheral point. To speed up the system response, redundant information like radius and angles are also stored for these element types.

Polygons connect data points by straight line segments. Splines connect data points by third-order interpolating curves with respect to a set of possible boundary conditions and damping associated with each data point.

On this level the ME Series 5/10 data structure for geometry can be viewed geometrically as a set of elementary curves with model points as geometric parameters. Topologically it can be viewed as an Euler net.

Contours and Faces. With ME Series 5/10 it is also possible to have composite geometric entities like contours and faces. This is an important feature supporting modeling. Contours are parameterized by the individual elements forming the contour. Faces are parameterized by one outer and no, one, or more inner contours. Parameterization means association: the changed underlying elements automatically influence all higher-dimensional elements.

Entities on a given level (>0) refer directly to entities on the level below. The problem of finding all elements on higher levels that refer to a given element is solved by a search process using the use count information. Therefore it is necessary to have use counters on all levels (<3). For instance, a use counter on level 2 indicates how many faces share a given contour.

Parts. It is desirable to collect all these element types in a single unit called a part. Then it is necessary to have a part element type (see the section on parts).

Splitting. In a technical drawing, elements such as lines and circles are drawn in a common plane. These elements can be images of physical edges in different planes. Suppose now that the drawing is a front view of a simple three-dimensional object consisting of two adjacent rectangles. Now add a horizontal line to the lower rectangle extending from the left to right boundary (see Fig. 2a). This new line may be the image of an edge in the same plane as the lines from the lower rectangle or it may belong to another plane (see Fig. 2b).

In the first case, the original rectangle should be split by the new horizontal line, in the other case not. How can the program know whether to split or not? The splitting function in ME Series 5/10 sets a flag. If this flag is true, at the insertion of a new simple geometric element like a line the system splits all elements passing through the endpoints of the new element automatically. If the flag value is false nothing happens. In this way the user can decide whether to split or not.

There may be situations where splitting a set of elements or splitting at intermediate intersection points is necessary. In this case, the SPLIT function can be used. The SPLIT function uses the whole selection mechanism, that is, the user can select a single element, or several individually identified elements, or all elements in a rectangular box, and so on. Then, in the case of a single selected element, the user is asked where to split. The CATCH mechanism can be used to indicate the split point. In the case of several selected elements the program calculates the intersection points between these elements and splits corresponding elements at their intersections (split_list).

Let us now describe how the split_list algorithm works. Suppose we are given a list of N elements. To find all intersections between these N elements one can consider all pairs (element i, element j), where $1 \leq i$ and $j \leq N$, and test these pairs for intersection. Because of symmetry, there are $N(N-1)/2$ of these pairs. So this algorithm behaves

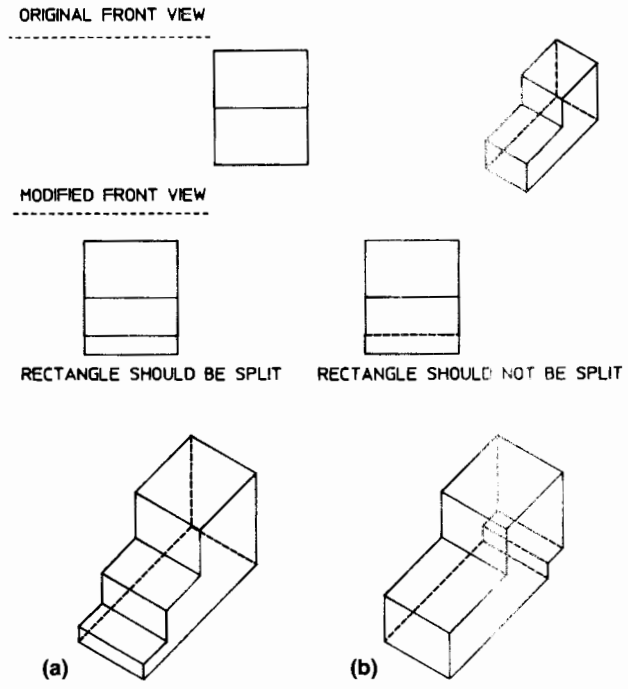

Fig. 2. *Two-dimensional representations of three-dimensional objects mean that the addition of a line can be the image of an edge in the display plane (a) or it may belong to another physical plane (b).*

quadratically, which means that splitting large element sets will be slow. With N = 1000 we would have to test for approximately $1000^2/2 = 500{,}000$ intersections. Assuming 500 intersections per second, this would take 1000 seconds.

In ME Series 5/10 another algorithm is used. As the first preprocessing step, the smallest rectangular axis-parallel box around the list elements is calculated. This box is divided symmetrically into four disjoint subboxes. In the second preprocessing step, the list elements are classified as either belonging to one of the four subboxes or extending across two or more subboxes. Clearly elements belonging to different subboxes can have no intersections. The test to determine if an element belongs to a subbox uses four real comparisons. Hence, the number of preprocessing steps is proportional to N.

Next, those elements that extend over several subboxes must be intersected with the corresponding elements. So an element extending over subboxes 1 and 2 may intersect

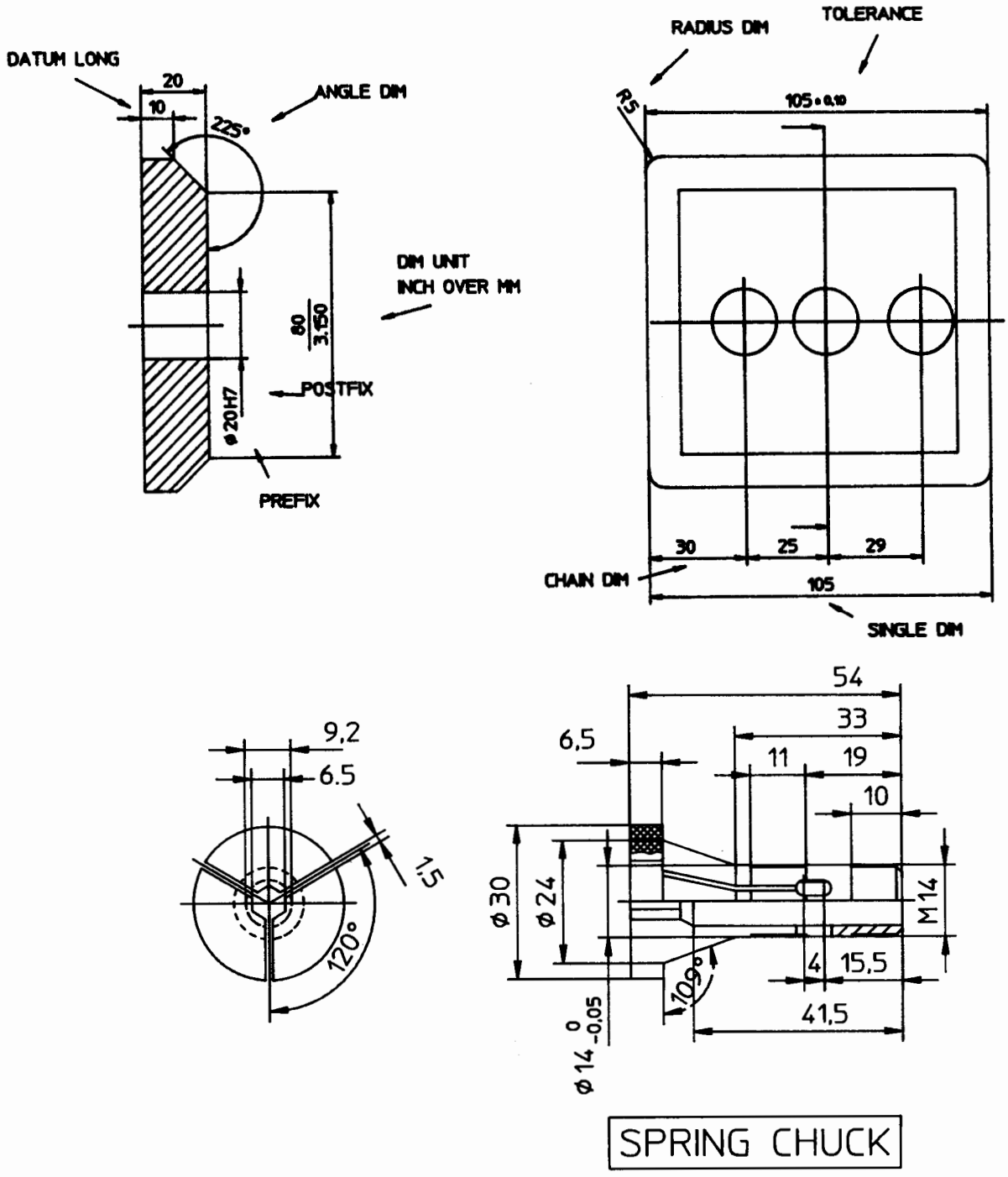

Fig. 3. *Examples of ME Series 5/10 dimensioning.*

elements in subbox 1 or subbox 2 or another extending element. Usually there are only a few elements extending over several boxes in a technical drawing.

Now we are left with four problems of the same type as the original problem. Suppose the distribution of elements is uniform in the plane and that there are no extending elements. Then each of the four subproblems has N/4 elements. The number of intersection calculations is now given approximately by $4(N/4)^2 = N^2/4$.

Now we can repeat the steps described above recursively. With every recursion level, the number of operations is decreased by a factor of 4 under the hypothesis given above. In the program an empirically selected constant is used to stop recursion. If the number of elements is less than this constant, the quadratic intersection algorithm is used; otherwise, recursion continues. So the actual behavior of this algorithm depends on the distribution of elements, the number of subbox extending elements, and the cost of preprocessing steps (on every recursion level).

Coordinates and Mapping

In vector algebra it is common to represent a planar point p in the form $p = O + x\mathbf{e}_1 + y\mathbf{e}_2$, where $(O,\mathbf{e}_1,\mathbf{e}_2)$ is a reference system consisting of a reference point O and two independent planar vectors $\mathbf{e}_1$ and $\mathbf{e}_2$. With respect to $(O,\mathbf{e}_1,\mathbf{e}_2)$ the point p has coordinates x,y, that is, p = (x,y).

Points need to be transformed in two ways. The first is to change the reference system and then express a given point in the new system. The second is to change a given point in a fixed reference system according to some type of mapping (move, rotate, etc). The latter transformation occurs in the MODIFY command. The former occurs if geometric information from one part is compared with that of a second part. Expressing both actions mathematically, the transformation of a point p to the new point p′ is given by:

$$p' = Ap + \mathbf{b}$$

where **A** is a 2×2 matrix and **b** is a translation vector:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Using homogeneous coordinates, $p' = Ap_h$, where $p_h = (x,y,1)$ and **A** is a 2×3 matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \end{bmatrix}$$

To avoid unnecessary arithmetic operations and to optimize certain error handling situations, a control mechanism was established. In ME Series 5/10 there is a type called geo operator, which consists of a matrix, a classification, and a flag. Classification of a matrix is according to its geometrical meaning:

- Unit (does nothing, usually used to initialize)
- Move only (two-dimensional matrix is unit, adds only)
- Orthogonal (preserves length)
- Conform (preserves shape, includes uniform scaling and orthogonal transforms)
- Affine (preserves parallelism, includes scaling along an axis and all former types)
- Perspective.

The flag indicates that the given matrix preserves orientation or reverses orientation.

The usual matrix operations like composition, inverting,

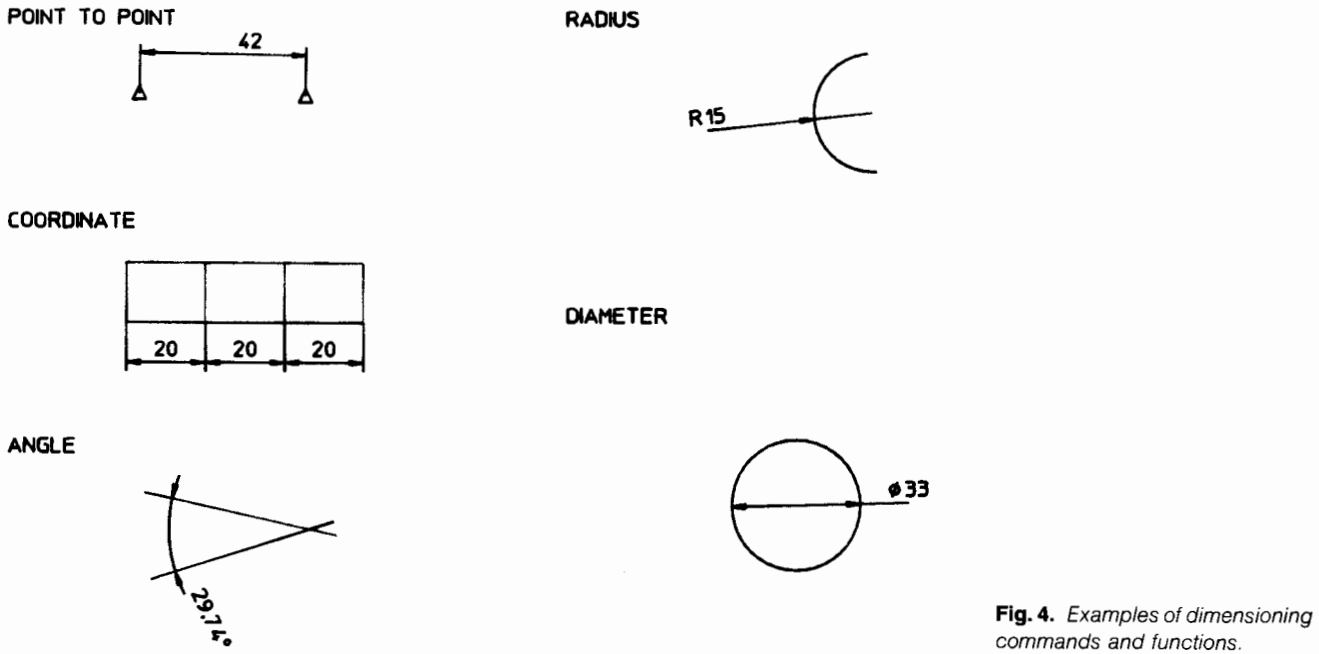

Fig. 4. *Examples of dimensioning commands and functions.*

and application to a point use the classification to optimize the action. For example, the inverse of an orthogonal matrix is the transposed matrix, which is obtained without multiplication.

The MODIFY and STRETCH commands in ME Series 5/10 use the same set of options to define the geometry of a modification. The basic options are MOVE, ROTATE, SCALE, SIMILAR, and AFFINE. There are suboptions for MOVE, ROTATE, and SCALE. With each modification the options DEL OLD or COPY and a repeat factor can be given.

These commands allow the user to define what should be modified, and how it should be modified. The "what" part is done by the general SELECTION mechanism for MODIFY and by a special selection routine for STRETCH. For the "how" part the user can define translations, rotations, reflections, and scaling in different ways. SIMILAR allows the user to define a transformation built up from a translation, a rotation, and a scaling operation by indicating two points and the corresponding image points. Affine mappings can be defined by three noncollinear points and their image points. Affine mappings may destroy the type of an element, for example, a circle will be transformed into an ellipse by the affine mapping

$$(0,0),(1,0),(0,1) \rightsquigarrow (0,0),(2,0),(0,0.5).$$

Because ME Series 5/10 in its current version does not support ellipse elements, the circle element will be converted automatically to a spline element in the geometric form of the desired ellipse.

The input to the general transformation algorithm used for MODIFY and STRETCH is a list of selected elements and a geo operator. For each element type there is a special modification routine. Modification of model points simply means multiplying the matrix by the coordinates of the model point. Modification of simple geometric elements like lines or arcs means point by point multiplication. So a line is transformed if its points are transformed. Contours are transformed if all constituent elements are transformed. Faces are transformed if their contours are transformed. Parts are transformed if their underlying elements are transformed. This can be done for shared parts by composing the shared part's matrix with the modification matrix. For nonshared parts the part's elements are transformed individually by recursion. (For details of the mapping of faces, see the section on hatching.)

Often two or more simple elements share a model point. If the image of such a model point is computed for the first time, the address of the image point is stored for further use in the temporary attribute associated with each model point. To use this information successfully, the temporary attributes of all model points that can be addressed by the given elements are set to an initial value, say NIL. Then, at modification time, an element is read from the list and processed depending on its type. The basic modification step is to apply the given geo operator to the model points associated with an element. If the value of the temporary attribute of the model point is NIL, the matrix of the geo operator is applied to the coordinates. This gives the coordinates of the image point. Now we need a new model point. So the coordinates of the image point are sent to the model point quad tree. If a model point with these coordinates already exists, the address of this model point is returned. Otherwise a new model point with these coordinates is created. The address of the new model point is then stored in the temporary attribute. The image model point is passed to the element. This basic modification mechanism for model points is the same for modification with copy or modification of the original element. In the case of copy, a new component record is created containing the new geometric (point) information.

To describe how STRETCH works, suppose the user has created a line and wants to stretch the right end of the line by ten millimeters. After calling STRETCH, the user indicates the line endpoint, places the ruler* parallel to the line, and indicates two ruler divisions on the ruler's x-axis separated by ten millimeters. On the program level, a line element is selected, the endpoint not to be modified is marked in the temporary attribute with its own address, the other endpoint (indicated by the user) is marked with NIL, and the translation operator is calculated according to the entered points. At modification time, the program changes the original 10-mm-translation geo operator to a geo operator that fixes model point 1 and shifts model point 2 by 10 mm along the line. To do so, the program reads the information in the temporary attribute of the model points. The new mapping is defined by $1 \rightsquigarrow 1$, $2 \rightsquigarrow 2'$, where point $2'$ is calculated from point 2 by application of the original geo operator. This type of mapping is a similarity operation. The new geo operator and the line element are passed to the transformation mechanism. So on the program level, stretching consists of two preprocessing steps: topological initialization of the temporary attributes and the calculation of a special geo operator for each (stretch) component from the global input geo operator. After these preprocessing steps, the normal modification algorithm applies.

*The ruler facility in ME Series 5/10 closely resembles a T-square. It was created to make the system resemble the traditional drawing board that most designers and drafting personnel are familiar with.

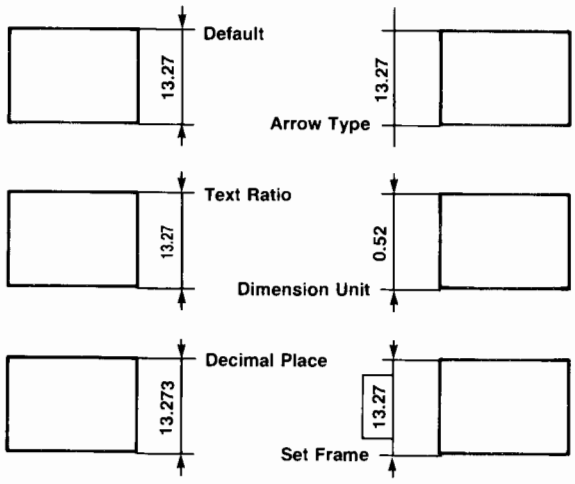

Fig. 5. *Dimensioning parameters.*

Scalar Product Geometry

The usual geometric calculations used in a two-dimensional CAD environment are:

- Length and angle calculations
- Orthogonal and parallel projections
- Parallels
- Intersections
- Tangential problems.

The basic objects to be dealt with are two-dimensional points and vectors considered to be elements of a real two-dimensional scalar product space. The usual Euclidean distance and the angle between vectors can be expressed by a scalar product <,>.

Two principles are used to perform geometric calculations:

1. Use a coordinate system related to the special structure of the problem.
2. Use an orthogonal coordinate system.

As an example, consider the ME Series 5/10 solution of the class of tangential problems called Apollonian problems. Consider all triples of elements, where an element is a point, or a line, or a circle. Given such a triple, determine the circle tangential to these elements.

There are ten special tangent problems, namely:

(A1)—point point point (point point radius)
(A2)—line point point (line point radius)
(A3)—circle point point (circle point radius)
(A4)—line line point (line line radius)
(A5)—line circle point (line circle radius)
(A6)—circle circle point (circle circle radius)
(A7)—line line line
(A8)—line line circle
(A9)—line circle circle
(A10)—circle circle circle.

A point can be considered as a circle with radius 0, so the class of different tangential problems involving elements and points can be reduced to:

(C1)—line line line
(C2)—circle circle circle
(C3)—line circle circle
(C4)—line line circle.

There also must be algorithms for the radius problems:

(C5)—line line radius
(C6)—circle circle radius
(C7)—circle line radius.

To distinguish between the different problems in the user interface, the following options were introduced:

- tan3, covering cases A7 through A10
- tan2_pt, covering case A4, A5, and A6 (radius input for pt is possible)
- tan_pt_pt, covering cases A2, A3, and A4 (radius input for pt is possible)
- three_pts, covering case A1
- center, covering the case where the tangential circle is given by its center and one tangential element.

Now, let us consider the solution to problem A10. Given three circles with centers m1, m2, and m3, and radii r1, r2, and r3, respectively, we want to calculate the center m0 and the radius r0 of the tangential circle. The tangential condition for (m1,r1) is $<m0-m1,m0-m1> = (r0 \pm r1)^2$ and likewise for (m2,r2) and (m3,r3).

In general, there are eight possible solutions to this problem. One possible way is to calculate all solutions, compute the deviations of the actual tangent points from the approximate tangent points and return the minimum deviation solution. In ME Series 5/10 preselection of the solution is performed by analyzing the approximate tangent points and temporarily attaching a sign to the radii r1, r2, and r3:

$$m0 = m1 + \lambda \mathbf{d_{12}} + \mu \times \text{Complement}(\mathbf{d_{12}})$$

where $\mathbf{d_{12}}$ is the unit vector from m1 to m2 and Complement($\mathbf{d_{12}}$) is $\mathbf{d_{12}}$'s orthogonal complement. If m1 = m2 = m3, then there is no solution except that also r1 = r2 = r3, so after a possible rearrangement we can assume $m1 \neq m2$. Inserting the unknowns, λ, μ, and r0, in the above tangent conditions leads to three coupled quadratic equations.

Decompose the vector $\mathbf{d_{13}}$ pointing from m1 to m3 into $\mathbf{d_{12}}$ and its complement: $\mathbf{d_{13}} = \tau \mathbf{d_{12}} + \kappa \times \text{Complement}(\mathbf{d_{12}})$. We then get the following system of equations:

$$\lambda^2 + \mu^2 = (r0 + r1)^2 \quad (1)$$

$$(\mathbf{d_{12}} - \lambda)^2 + \mu^2 = (r0 + r2)^2 \quad (2)$$

$$(\tau - \lambda)^2 + (\kappa - \mu)^2 = (r0 + r3)^2 \quad (3)$$

These equations are solved by standard elimination techniques. There are two solution branches depending on whether $\kappa = 0$ or $\kappa \neq 0$. $\kappa = 0$ means geometrically that the three centers m1, m2, and m3 are colinear. In this case the radius r0 can be found by solving a system of two linear equations. If $\kappa \neq 0$, then r0 is the solution of a quadratic equation.

Geometry Software Architecture

The basic software module in ME Series 5/10 for the

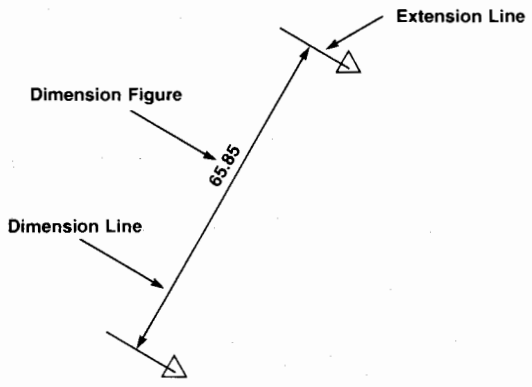

Fig. 6. *Example of point-to-point dimensioning.*

creation of geometry is called ACT_GEO. For each type of simple geometric element, a specific action routine in ACT_GEO lets the user create elements of this type in various ways. To create a circle tangential to three other circles, the c_circle routine in ACT_GEO calls a specific routine in CMPNT_CIR. This routine in turn calls a routine in SOLVE to calculate the center and the radius of the tangential circle. If a solution is obtained, another routine in CMPNT_CIR checks the solution, and if the check is positive, calls a routine in MODEL_DS to create a circle element and then calls a routine in DRAW to draw the circle onto the graphics area. Besides ACT_GEO, the software modules in ME Series 5/10 are:

U_GLBLS contains the basic definitions of constants such as π, data types such as points, etc.

GEO_MTR contains the routines for scalar product geometry.

SOLVE contains the equation solver routines for all the geometric problems ME Series 5/10 can handle.

MIN_DIST contains routines that calculate the distance and the projections from points to elements. These routines are used by CMPNT_ANY and MODEL_ACC to identify elements by a pick in the graphics area.

MAP_GEO contains the calculus to deal with geo operators. Routines from this module are used by part routines, modification routines, and routines associated with the user coordinate system.

MODEL_DS contains insert, get, change, and delete routines for elements realized as abstract data types in a standard format.

CMPNT_ANY contains routines that perform element type independent calculations such as:

Routines to get other geometric representations of elements (e.g., circle data from an arc).

Distance from point to element.

Tangent vector at a point to an element.

Intersection point(s) between two elements.

Approximation of an arbitrary element by a point polygon.

MODEL_ACC contains all the search routines, using MODEL_DS and CMPNT_ANY.

CATCH uses the MODEL_ACC routines to catch according to the catch mode.

MODEL_CHG contains split, merge, delete, etc. subroutines.

CMPNT_LIN, CMPNT_CIR, CMPNT_FIL, etc. contain the high-level routines to create elements.

There are two levels in this software structure with respect to data structure. On the low level, objects such as constants, numbers, points, and vectors are defined and can be exchanged between routines. On the high level, beginning with MODEL_DS, one deals basically with abstract objects such as faces, contours, lines, and model points. For example, to catch to the intersection point of two lines on the screen, the user indicates this point approximately by a pick with the graphic cursor. Then CATCH calls routines in MODEL_ACC that identify the part, part matrix, and line elements of the intersection lines. The intersection routine in CMPNT_ANY gets all this information and decomposes the line elements into beginning and ending points, which are transformed by the part matrix to a suitable common coordinate system. This point information is then passed to a routine in SOLVE that calculates the intersection point relative to the coordinate system. After transformation of the intersection point back to the world coordinate system, the MODEL_ACC routine checks to see if the intersection point is in the range of the graphic cursor. If this is not the case, the CATCH routine passes a NOT FOUND to the calling action routine. If there are more than two elements passing through the cursor range, then all possible intersections

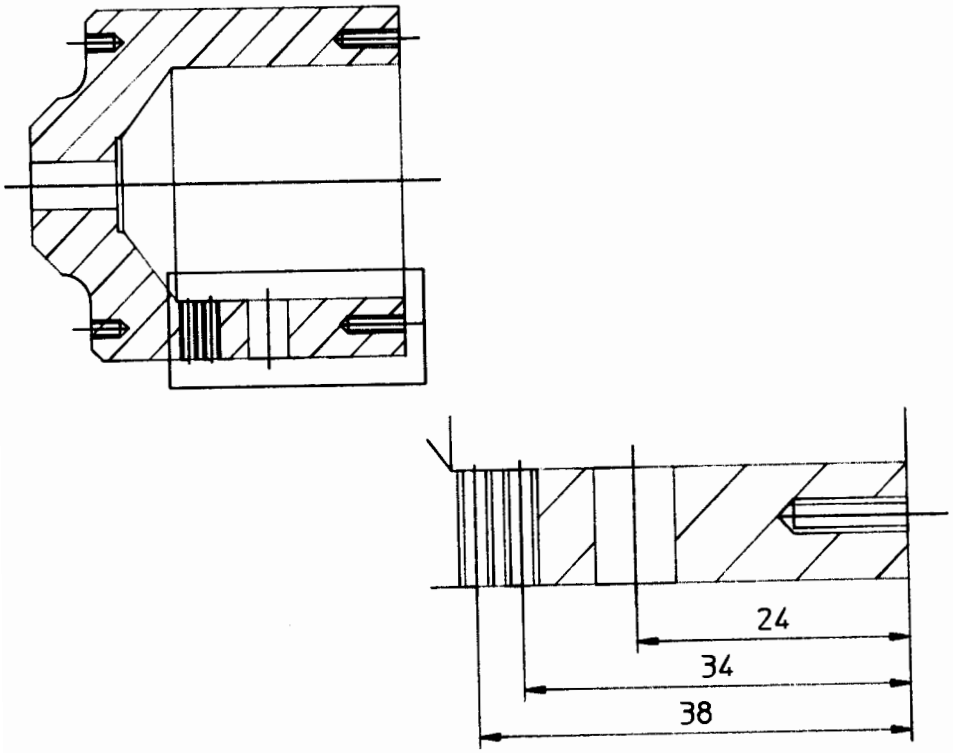

Fig. 7. *Example of the result of the* DETAIL *command.*

are calculated and the closest to the indicated point is chosen.

Dimensioning

Dimensioning a given mechanical part means to give a description of the physical size, shape, and position of the geometry of that part in a human readable form. Hence, dimensioning is one of the most important operations in creating a detailed technical drawing (Fig. 3). There are national and international standards for dimensioning: ANSI, ISO, DIN, and JIS, for example.

CAD User Expectations. CAD systems should help the user concentrate on the design process. Dimensioning a part in a technical drawing should be easy and fast. The system should provide a complete set of dimensioning functions. It should be possible to dimension according to a given standard. Often it happens that the geometry is changed after dimensioning. In this case dimensioning should be associative, that is, follow the modified geometry automatically. This system behavior saves a lot of work and time for the designer. Changing of dimensioning should be easy. Sometimes it is important that a system be able to perform isometric dimensioning. For factories using both metric and nonmetric dimensions it may be important to have a dual dimensioning capability.

There are many routine tasks: conversion of units, placing of dimension and extension lines, spacing checks, changing the height of a dimension text for microfilm purposes, etc. It is expected that a CAD system will perform these tasks with as little as possible user interaction.

Dimensioning in ME Series 5/10. One of the ME Series 5/10 design principles is fidelity of the created data, that is, the user creates geometry that corresponds to the physical size of the part that will be manufactured. Hence, dimensioning a part in ME Series 5/10 means to make the inherent geometrical properties of the part visible in human readable form.

The following dimensioning commands and functions are implemented (see Fig. 4):

- Line dimensioning. The distance between two points or the length of an edge can be represented by either datum or chain dimensioning. Horizontal, vertical, or parallel dimensioning is optional. Coordinate dimensioning is possible. To create a dimensioning line, geometrically defined points such as the endpoints of elements or the centers of circles are needed.
- Radius or diameter dimensioning.
- Arc dimensioning.
- Angle dimensioning. Suppose a user wants to dimension the angle between two lines. By digitizing the text position in one of the four possible quadrants defined by the lines, it is possible to choose the angle of interest. The dimension figures can be expressed in decimal as well as in degree-minute-second notation.

The shape of a dimensioning can be optimized by the user by changing one or more of the following parameters (see Fig. 5):

- Text standard used
- Height and width of dimension figures
- Direction of dimension figures
- Gaps between edge and extension lines
- Length of extension lines over the dimension lines
- Number format. For example, the number 0.100 can be expressed as 0.100, or 0,100, or .100, or 0,1, and so on.
- Position of dimension figures with respect to the dimension line
- Units (mm, inch, fractional dimensioning in foot and inch, dual dimensioning).

Data Structure. There are three design principles for the dimensioning data structure: flexibility, associativity, and compactness. Flexibility means including a complete set of dimension parameters in the data structure so that a user is able to generate dimensions according to a particular standard. Associativity means using references to geometry

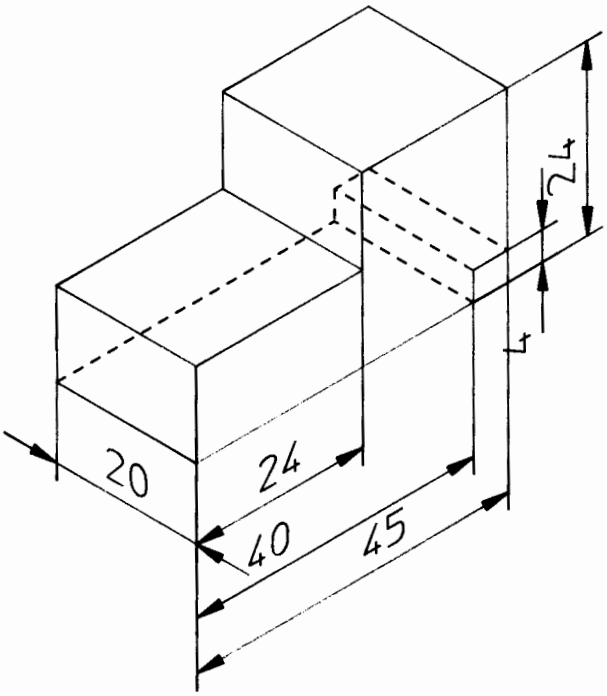

Fig. 8. *Isometric dimensioning can be given to shared parts only.*

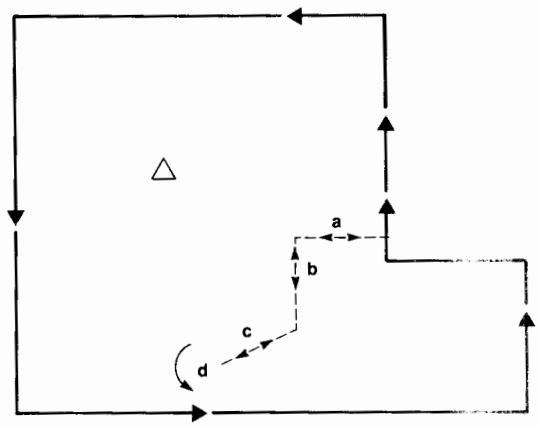

Fig. 9. *Finding the closed outer contour (solid line). The algorithm backs out of "dead-end streets" like abcd (dashed line).*

to let the dimensioning automatically follow any modification of the geometry. Compactness means to store only the most basic pieces of an actual dimensioning so that it is possible to reconstruct the expected display image from the stored data.

As an example, we consider a point-to-point dimensioning (see Fig. 6). The picture on the display consists of the geometry, the dimension line, the extension line, and the dimension figure. Stored in the data structure are references to the geometry, the values of parameters, and the dimension text location in relative coordinates with respect to the geometry. Drawing this dimensioning means adding the nonstored dimension data by using an algorithm so that the complete display picture appears.

What do relative coordinates mean? In the case of our example, the user must indicate two vertex points and the text position. The text position point is transformed into the coordinate system where the origin is defined by the first-indicated point, the x-axis by the line connecting the geometry points, and the y-axis by the line parallel to the extension lines passing through the origin. The length unit on the x-axis is defined by the distance of the vertex points; the length unit on the y-axis is defined by the internal units (mm). The information on this user-defined text location is kept even if the text figure does not fit between the extension lines. In such a case, the dimension figure will be drawn outside the extension lines. If, after a modification, the text figure will fit, the system remembers the text position and draws the text at the user-defined position.

To make the data structure more compact, a set of parameters that usually have equal values across a drawing is kept in a special buffer as a single item. As long as these parameters are used, any newly created dimensioning points to this item in the buffer. If one of the parameters is changed, the system searches in the buffer for an identical item. If such an item is found it will be used; otherwise, a corresponding item will be generated. The buffer is organized as a linear list.

Dimensioning is attached to a part. A part is an element that contains other elements of any type, including parts.

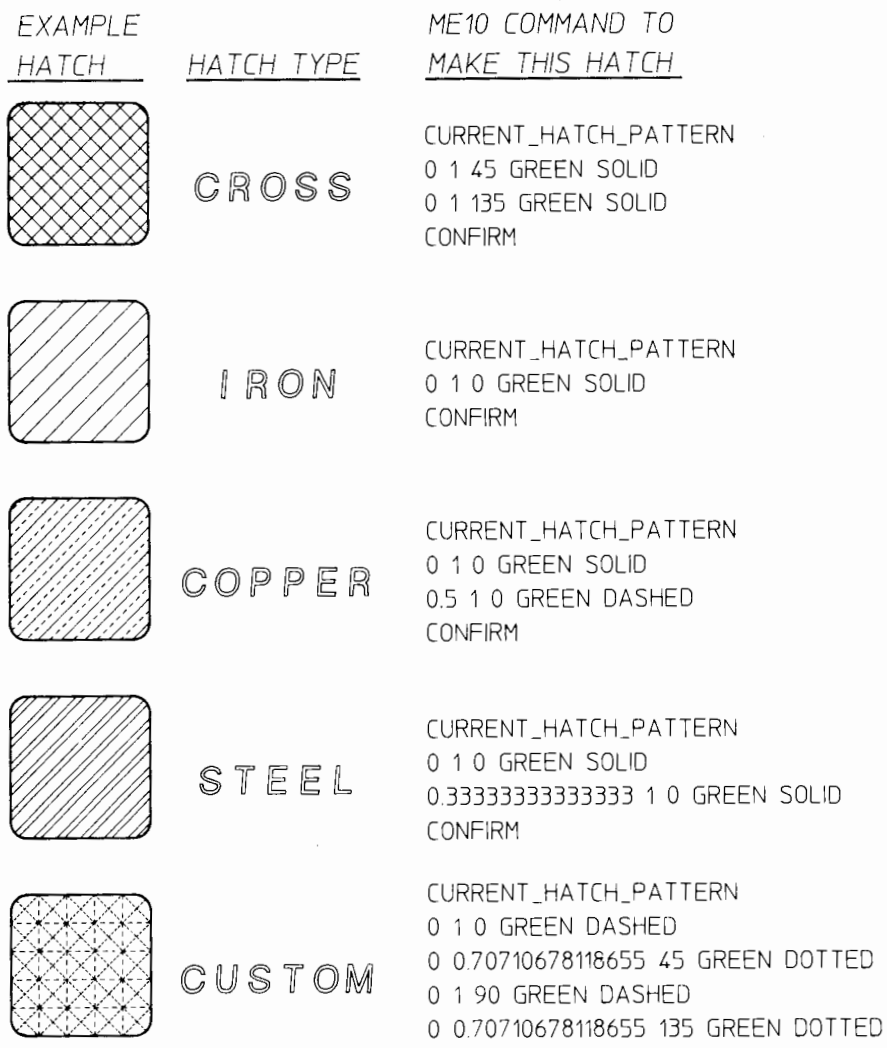

Fig. 10. *Examples of hatching. Hatch defines the location, orientation, and size of the pattern on the face.*

The dimension data is stored in this part.

Associativity of dimension data is an important feature in ME Series 5/10. Therefore, the dimensioning algorithms follow a high-level description of the algorithms invoked by the MODIFY command. The user selects geometry elements to be modified. These elements are kept in a separate list. The system searches the dimensioning data pointing to geometry in this list and collects the data for the selected elements in a separate dimensioning list. The geometry is then modified sequentially and the affected dimensioning is automatically adjusted to the values of the new geometry.

The text location of a dimension is defined with respect to a mixed coordinate system. One axis of the coordinate system (the x axis) is changed automatically by the new geometry values, but the other axis value must be calculated from the transformation type and the old value. In addition, for stretching, the transformation can change from element to element. So the dimensioning must be modified in parallel with the geometry.

Parts in ME Series 5/10 have their own transformation (see section on parts data structures). The purpose of this transformation is to express the part content in the parent coordinate system. This transformation is classified by its geometric meaning. In addition to the mathematical meaning (length, angle, parallelism preserving), there is information on the use of the part data. In the case of an isometric view, the part data does not represent physical data, but instead a view of the physical data. Hence, in ME Series 5/10 an additional part identifier is included to cover the cases of modeltype, viewtype, and detailtype parts.

By default, the system assigns a modeltype identifier to a new part. Using the ISOMETRIC command with shared parts, the system automatically assigns the viewtype identifier. On the other hand, the result of the DETAIL command is a shared detailtype part (see Fig. 7).

The part identifier is used to draw the part dimensioning. The display appearance of a dimension is generated by a mapping mechanism, converting the stored data to the display data. In the case of a modeltype identifier, the geometry information is transformed by the part transformation, and then the normal display mechanism applies to this data. For a detailtype identifier, the system behaves as in the modeltype case except that the dimension figure is calculated from the original values. In the case of a viewtype identifier, the geometry information remains unchanged while the display mechanism is adjusted to include the part transformation. In the current state of the program, an isometric dimensioning can be given to shared parts only (Fig. 8).

```
INIT_SUBPART 'RIGHT HOLE'
  LINE . . .
  ARC . . .
END_PART

LINE . . .
ARC . . .

CREATE_SUBPART 'LEFT HOLE'
  [IDENTIFY THE LINE + ARC]
```

Fig. 12. *List of commands that could have been used to create the drawing shown in Fig. 11.*

Hatching

Besides the geometric information, the ME Series 5/10 data structure also contains topological information. The data structure can be viewed as a topological net. A topological net is a finite set of objects from two abstract classes called knots and edges obeying the following rules: an edge connects two knots (equal or unequal) and at least one edge extends from each knot.

The elements of the topological model are:

- Model points: the knots of the topological net.
- Primitive components: edges starting and ending at model points (e.g., lines, circles, circular arcs, splines, and polygons).
- Contours: an ordered set of primitive connected components where every component occurs only once in the contour.
- Faces: defined by one outer and none, one, or more inner contours.

Primitive components are connected if they share at least

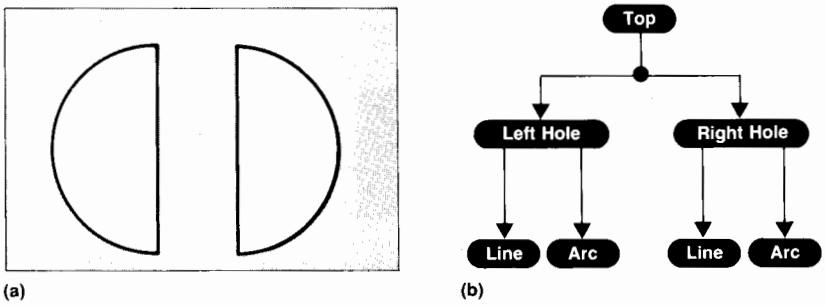

Fig. 11. *Simple ME Series 10 part drawing. (a) Geometry. (b) Parts structure.*

one model point. The components of a contour are ordered so that every component except the first and the last is connected with both of its neighboring components. A property of the contour is that the components are connected with no other components in the contour except their neighboring components. If the first and the last component of a contour are connected, the contour is called closed.

Recognition of Faces in Topological Nets

ME Series 5/10 includes a mechanism that recognizes and creates faces identified by the user. There are various ways in which a user can identify faces to the system. The easiest is to indicate a point on the face. Because faces are defined by their boundaries, there are two tasks: finding the closed outer contour and finding any closed inner contours.

Finding the Closed Outer Contour. An infinite ray starting at the point indicated intersects at least once with every contour that includes the point. Starting with the component the ray intersects first, the algorithm tries to find a closed contour including the point. If no such contour can be found, the next component intersected by the ray is used for the search.

The beginning and ending points of the start component for the contour search are defined so that the component is run through counterclockwise relative to the point that has been indicated. Starting at the endpoint we search for connected components until the beginning point of the start component is reached. If a point is reached where we have the choice between several components to go along, we use the one leading to the left. Since we run around the point in a counterclockwise sense, going left ensures that we find the innermost contour of all the contours that include the point.

If a point is reached where no connected component can be found, we change direction and go back the same way until we come to the last point where we had the choice between several components. There we take the component leading to the left. This is the one we would have taken if the dead-end street we just took was not there.

Only two rules are used to determine the connected components. The first tells what to do when reaching a knot that has several branches, and the second tells what to do when reaching an endpoint of the net.

Keeping track of the route results in a list of components where preceding components are connected by a common point. This list contains all the dead-end streets we tried, so they are to be sorted out. Every dead-end street forces a return. In the list a return is documented by a component followed by itself. If we go back along a dead-end street, we pass the same components we passed on our way in but in opposite order. Hence, every dead-end street has the following appearance in the component list: ...a-b-c-d-d-c-b-a.... (see Fig. 9).

Removing the return component d followed by itself yields: ...a-b-c-c-b-a... Going on to remove each component followed by itself results in a contour with no dead-end streets. It is obvious that the algorithm not only works with simple dead-end streets but also with complex connections of dead-end streets.

Removing all dead-end streets may give a closed contour as we have defined it above, but it need not. It also may result in a few closed contours connected by bridges consisting of one or more components. Bridges exist if we still have components in our list that occur at least twice but are separated by other components. If we split our list at these bridges we get several closed contours. One of them is the outer contour we are searching for, the others are inner contours connected with the outer contour. To select the outer contour we search for the contour inside which the indicated point lies.

This test depends on the fact that a point lies in a contour when the number of intersections of any ray starting at the point with the contour is odd. Since the direction of the ray is optional, it is a good idea to choose a vertical or horizontal ray to make computation easy and quick.

Finding the Closed Inner Contours. First, the number of components that are used for the search of inner contours is reduced by a box test using the box of the outer contour. Then starting with an arbitrary component, we try to find contours by searching for connected components. If a point is reached where more than two components are connected, all components reachable from that point are recursively searched. Thus, a subset of components is gained. This subset forms a net. The outline contour or contours of the net are to be found. This can be done using the same algorithm described above for finding the face's outer contour. The only difference is that we now use a point outside the contour we hope to find.

The start component for the outline search is the component reached first when an arbitrary ray from a point outside is shot through the net. The beginning and ending points of the component are ordered the same way as above. The turn-left rule used for finding the outer contour of the face led to the innermost contour around the point. Since

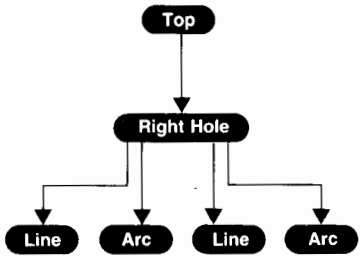

Fig. 13. *Desired modification of Fig. 12 parts structure (see text).*

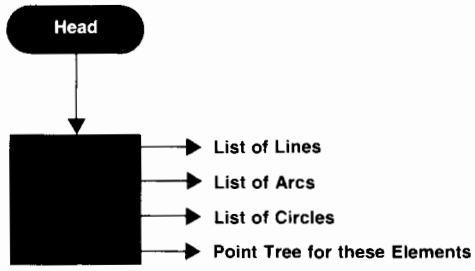

Fig. 14. *Graphical representation of a part data structure in ME Series 10.*

the point now lies outside the net, the turn-left rule leads to the outermost contour of the net. If only points occur connecting two components, the contour is found when the beginning point of the first component is reached again.

When all contours in the box of the outer contour are found, we have to sort out those that do not lie in the outer contour and those that lie in other inner contours. The following rule is used: *A contour I lies in a contour O if I does not intersect O and one point of I lies inside O.*

Hatch Data Structure

In mechanical engineering, the exposed cut surfaces of sectional views are indicated by hatching. Symbolic hatching is used to distinguish various materials. In ME Series 5/10, symbols for materials can be created by superimposing sets of parallel lines. The symbols are called patterns.

The data structure for the hatch in ME Series 5/10 consists of three elements:

- Simple pattern—all the data needed to define a set of parallel lines. It contains display information like color or linetype and geometric information defining the distance between the lines and the orientation and location of the line set in the pattern.
- Pattern—a simple pattern or a connection of simple patterns.
- Hatch—defines the location, orientation, and size of the pattern on the face (see Fig. 10).

A pattern can be shared by hatches, and a hatch can be shared by faces.

In many CAD systems, hatch is a separate element not coupled with geometry. In ME Series 5/10 another approach has been chosen. Hatch is considered to be an attribute of faces. Hence, hatch is coupled with geometry through faces.

Modification of Faces and Contours

The composite components, contours and faces, are updated automatically when the underlying simple subcomponents are modified. However, in the case of a MODIFY copy operation, hatches are copied automatically. The user cannot select faces and contours directly. Selection works on the simple component level. So the generic input for the MODIFY command consists of a list of simple components. This means that faces and contours pointing to these simple components have to be recognized. If all the simple components of a contour are in the list, the simple components are replaced by the contour. If all the contours of a face are in the list, the contours are replaced by the face. The result of this preprocessing is a list containing parts, faces, contours, and independent simple components.

In ME Series 5/10, composite components can share their subcomponents. So we must take care that a component belonging to several composite components in the list is not modified twice or more. Therefore, the first step of the modification algorithm is to initialize the temporary attribute associated with each component. Only if the value of the temporary attribute of a component has its initial value at modification time will this component be modified and the address of the image component stored in the temporary attribute. The modify information stored in the temporary attributes is also used by an UNDO of the current MODIFY command where the inverse transformation is applied to all components with noninitial temporary attributes. This is especially important for the STRETCH command, since the original topological information about how to stretch is condensed to the temporary attributes.

Parts

Up to this point we have discussed a number of elements: lines, circles, arcs, and so on. Now we turn to a new element: parts. A part is an element that contains other elements of any type, including parts. A part usually corresponds to some physical object, like a screw, or to an assembly of physical objects, like a pump. But parts can be simply groups of elements that the user finds convenient to manipulate as a unit, and may have no direct physical meaning.

Parts in a drawing are like directories in a file system. A part is an element, just as a directory is a file, and a part contains elements, just as a directory contains files. A part can contain a part, just as a directory can contain another directory. Parts impose a hierarchy on the elements of a drawing, just as directories do with a file system.

It is common to represent this hierarchy graphically as a tree, much like a directory tree. Fig. 11 shows a simple ME Series 5/10 drawing representing two symmetrical holes, and the parts structure for this drawing. Elements are represented as circles. The elements on the bottom line are lines (L) and arcs (A); the other elements are parts. Each hole is a part consisting of a line and an arc. The two holes are named Left Hole (LH) and Right Hole (RH). The holes belong to a part called Top, which is present in every ME Series 5/10 drawing.

The two holes could also be drawn without parts, in which case the only elements of the drawing would be the two lines and the two arcs, plus the everpresent Top part. So why should a user bother to create the part structure in Fig. 11? There are several reasons:

1. A part can be treated as a single unit when modifying, deleting, storing, or loading. Assume the user wants to move the two holes closer together, for example. The user

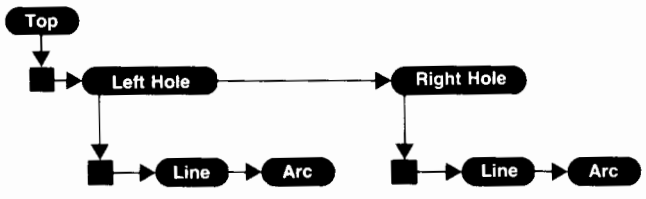

Fig. 15. *Data structure representation of part shown in Fig. 11.*

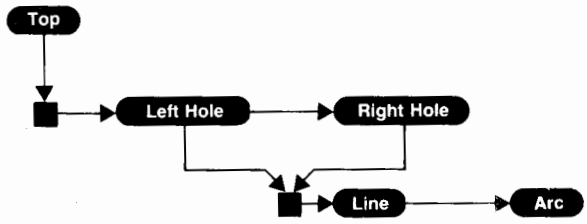

Fig. 16. *Different data structure for Fig. 11 if LH and RH were shared.*

can identify the part to be moved by picking any element belonging to it. With complex parts, this can be very convenient.

2. Parts can be reused. The user might keep a library of commonly used screws, for example, and load them as parts when needed. If a drawing is made completely from library parts, the user needn't actually draw anything; the user only needs to say where the parts go.

3. Parts can be shared. When two parts in a drawing are shared, any change to one is reflected immediately in the other. The two holes in Fig. 11 could have been created as shared parts, for example, although for simplicity they were not.

One goal of ME Series 5/10 is to treat parts like other elements as much as possible. We have tried to avoid creating two sets of commands, one for use with simple elements and the other for use with parts. Instead, we try to use the same commands for all elements. For example, there is no DELETEPART command; there is only DELETE, which applies to parts as well as other elements.

Nevertheless, there are a few special parts commands. Two commands are used to create parts. With CREATESUBPART, the user specifies which elements will belong to the new part with the requirement that the elements must already exist. INIT_SUBPART creates a new part with no elements; the user can then add the elements that will be in the part. Whether CREATE_SUBPART or INIT_SUBPART is used is the user's choice. The drawing and parts structure in Fig. 11 then could have been made with the commands listed in Fig. 12.

EDIT_PART chooses the active part, which is very much like the current directory of a file system. Only one part can be active at a time. The active part is the one to which all new elements belong. In Fig. 12, INIT_SUBPART both created a new part RH and changed the active part to be RH. LINE and ARC then created elements in RH.

The active part is also important for identification with the tablet or mouse. Suppose the user wants to delete RH. The user picks DELETE from the menu, and then picks, say, the arc of RH. How does the system know whether to delete just the arc, or the entire part? It doesn't, without some convention. The convention we have chosen is that the element to be deleted (or modified, or identified for some other purpose) is always in the active part. In our scenario, if the active part is Top, the system deletes RH. At that level, RH is an indivisible unit. But if the active part is RH, the system deletes the arc. What if the active part is LH? In this case, the system has no way to tell what the user wants, so it just beeps.

The active part is so important that we have done two things to make plain to the user where the part is. First, the status line contains its name. Second, elements in the active part are drawn with the color and linetype chosen by the user (default values are white and solid), while all other elements are drawn in magenta and dotted, making them less prominent on the screen.

Two commands are used to rearrange the parts structure: SMASH_SUBPART and GATHER. SMASH_SUBPART deletes a part, but leaves its member elements behind in the same place. GATHER pulls elements into the active part. As an example of the use of these two commands, suppose we want to modify the parts structure of Fig. 11 so that it looks like Fig. 13. That is, all four simple elements are to be members of the same part. One possible way would be to edit the Top part and then enter SMASH_SUBPART "LH". Now the drawing has the structure shown in Fig. 13, and the job is half done. To finish, we edit RH and then enter GATHER and identify the line and arc that used to belong to LH. This produces the result we want.

VIEW is a command that causes a part to be viewed in a viewport without its surrounding context. Normally, a port has a view of the Top part, but it is sometimes convenient to view some other part while you are working on it. Normally it suffices just to change the window, but if two parts overlap, only a view can show one without the other. It is even possible to have several viewports, with an overview of Top in one port and views of other parts in other ports.

Parts Data Structures. Internally, each element is represented by a block of memory containing the element's color, linetype, and other properties common to all elements. In the following discussion, we call this block the element's head, and represent it as a circle. At the end of the head is data specific to the type of element represented. A line, for example, has pointers to its endpoints. A construction circle has a radius and a pointer to its center point. A part has a pointer to a block we call the part's body. Graphically, we might represent a part as shown in Fig. 14. The part body contains lists of its member elements, one list per element type. Having one list per type makes the system faster in several situations than it would be if all elements were in a single list. These situations arise when the system knows that only elements of certain types are interesting for the command. CONVERT_SPLINE, for example, converts splines to lines and arcs, and needn't consider any elements other than splines. Its also possible for the user to qualify a command with an element type, as in DELETE CIRCLES ALL, which deletes circles.

Each part has its own point quad tree, containing the model points for all its member elements, except those that themselves are parts. This arrangement is much more suitable than having a single point tree. It is very simple to delete a part with the current scheme, for example. There is no need to restructure a large point tree; we just throw away the tree of the deleted part. This scheme is also convenient for point catching, since we can easily search for points in visible parts without having to look at points in

```
DRAW(ELEMENT, MATRIX)

FOR EACH NON_PART ELEMENT G
  DRAW_ELEMENT(G, MATRIX)

FOR EACH PART ELEMENT P
  M:= P'S MATRIX
  DRAW(P,MATRIX*M)
```

Fig. 17. *Simple version of a draw routine. The matrix represents a mirroring transformation.*

parts that are not being viewed.

The fact that a part's head is separate from its body is the key to shared parts. Fig. 15 shows a data structure representation of the drawing in Fig. 11. Notice that LH and RH contain different elements. One part can be changed independently of the other.

If LH and RH were shared, the display would look exactly the same, but the data structure would look like Fig. 16. The structure is much the same as that of Fig. 15, but with one essential difference: the two parts LH and RH share the same body, and thus the same elements. In this case, a change in one part is a change in the other, since internally there is only one representation of the two parts' contents.

How can a data structure with one line and one arc in Fig. 16 look the same as one with two lines and two arcs in Fig. 15? How does ME Series 5/10 know that the shared line and arc are to be drawn twice? How does it know where to draw them? The answer is that each part head contains a matrix telling how to transform the part contents for the part instance represented by the head.

The matrix can represent several types of transformations: none at all, a translation, a rotation, a scaling, or a mirroring. In Fig. 17, the matrix represents a mirroring. The matrix can also represent combinations of transformations as well as parallel perspectives. The matrix is created by the MODIFY command when it notices that the part has been declared shared by the command SHARE_PART. If the part is not shared, the MODIFY command must transform and possibly copy every element in the part. This can be slow for large parts. Hence, another advantage of shared parts is that the execution of MODIFY commands is much faster.

Since shared parts can be nested, there may be several matrices that apply to a given element. To draw an element, the system needs to transform it with each matrix along the path from that element to the Top part. To draw every element (when the user changes the window, for example), the draw routine must walk over the entire parts tree, maintaining at each point the product of matrices from the Top part to that point. A (very) simplified version of a draw routine is shown in Fig. 17 in a mixture of Pascal and English. The routine shows the sort of recursive approach that pervades all of the parts software.

Selection Mechanism

In many ME Series 5/10 commands, the first action is to select elements. Examples are DELETE, MODIFY, SPLIT, CHANGE_COLOR, CHANGE_FILLET, etc. For these commands, ME Series 5/10 uses a common mechanism called selection. In other commands, such as GATHER and STRETCH, more information has to be selected, so these commands use a different selection mechanism.

Selection works either locally in the current part or, if the user has entered GLOBAL, in all parts.

Implicit Selection. Suppose the user wants to delete only one line. Probably the easiest way to do this is to call DELETE and to pick this element on the screen. On the action routine level, the delete action routine passes control to the selection action routine. This routine finds a point token from the user pick on the screen. This piece of information causes a search of the data structure to find the closest element to this point within the search range. In this case the search is successful and the address of the line element is passed to the delete action routine. This is what is called implicit selection. The user poses an easy-to-solve problem and the system need not have more information to solve the problem.

Other features of the implicit selection are given by the automatic vertex and box option. In picking a vertex, the system selects all elements that share this vertex. Now suppose the user entered a point and no elements passing through the search circle around this point were found. Then the system interprets this point as a box corner and asks for the other (diagonal) box corner. Then the elements completely inside this box are searched and passed to the calling action routine.

Explicit Selection. To get explicit selection the user enters the SELECT command. For example, the user wants to delete three lines scattered around the screen. Probably the easiest way to do this is to use DELETE SELECT pick1 pick2 pick3 CONFIRM. SELECT triggers the selection action routine to expect input until the loop is finished with CONFIRM. As described above, a point token lets the selection routine search for a closest element. VERTEX and BOX can now be entered as explicit options.

With explicit selection the user can select elements with more options. There are the logical operators AND, OR, EXOR, NOT, ADD, and SUBTRACT. The element types POINTS, LINES, ARCS, FILLETS, CIRCLES, TEXTS, SPLINES, C_LINES, C_CIRCLES, PARTS, GEO, and C_GEO can be used. Colors and linetypes can also be used as search patterns. For example, it is possible to select all blue arcs included in box 1 and not included in box 2. The user can also select all elements with the associated information string "remove me" that are not of long-dotted linetype. In connection with information literals, wild cards can be used. SELECT INFO "*er" selects all elements whose information strings end with "er".

Reference

1. H. Samet, "The Quad tree and Related Hierachical Data Structures, *Computing Surveys*, Vol. 16, no. 2, June 1984.

Alpha Site Evaluation of ME Series 5/10

by Paul Harmon

WHEN WE FIRST HEARD at HP's Vancouver, Washington Division of a new software package being developed in one of HP's facilities in Germany for two-dimensional computer-aided drafting (2D CAD), there was skepticism about the value of developing such a product within HP. Several software packages of that general description already existed and some were very good. Within HP, HP Draft had just been released on HP 9000 Series 200 Computers, and EGS-200 was getting a major overhaul that greatly increased its performance as a mechanical engineer's tool. Outside HP, several other programs developed by other manufacturers were available. Beyond that, many felt that recently announced systems that incorporated three-dimensional CAD capability were much more powerful than any possible 2D system and, though many times more expensive, represented a better investment.

At HP's Vancouver Division (VCD), we had two projects that were making use of CAD to design next-generation personal workstation printers. The first project used HP Draft on our HP 9000 Series 200 Computers and was an alpha test site for the second generation of EGS-200. The second project used another company's three-dimensional solids software on dedicated hardware. Another group, still working with pencil and paper, was considering making the switch to CAD.

Consensus as to which system the whole lab at VCD should adopt was difficult to arrive at, since each option had advantages as well as disadvantages.

The Ideal CAD Workstation

As we worked with our CAD tools, we began to see what an ideal system should include:

- The hardware should be inexpensive so that each engineer can be provided with a personal station, eliminating timesharing (which we found to be deadly to productivity).
- The system should be powerful enough to execute the CAD program quickly while being flexible enough to use for other engineering tasks such as instrument control, custom analysis, document preparation, and electronic mail.
- The individual workstations should be networked so that data can be shared instantaneously within the development organization, and eventually between organizations.
- The operating system should be easy to comprehend and allow multitasking for shared tasks such as instrument control, long batch mode analysis, etc.
- The program must be accurate enough to cover the range of sizes we would have in a typical layout.
- The software should incorporate a three-dimensional solid modeler as well as an easily understood two-dimensional graphics editor with a full feature set.
- It should be possible to customize the user interface and write special subroutines, or macros.
- There should be links to both finite element (FE) analysis programs and numerically controlled (NC) machinery for CAD/CAM integration.
- Links should exist to outside vendors, possibly through the Initial Graphics Exchange Standard (IGES).
- There should be easy access to spreadsheet, word processing, and electronic mail to provide an integrated workstation for maximum productivity.
- Above all, each part of the system should be reliable. It doesn't take long to eliminate all productivity gains if the system fails often.

The CAD systems we were familiar with provided less than half these capabilities. When we asked about ME Series 10, we found that it was being designed to run under the HP-UX operating system (HP's version of AT&T's UNIX™ operating system) on HP desktop computers, provide links to FE and NC packages that we were already successfully using with HP Draft, and provide a macro language which would allow a high degree of flexibility. Although lacking three-dimensional solids modeling capability, ME Series 10 was always intended to be the user interface for the new three-dimensional modeling system (ME Series 30) introduced by HP in November 1986. Since the description sounded so close to our concept of the ideal CAD system, we offered to be an alpha test site. Although HP's Böblingen Engineering Operation in Germany (the authors of ME Series 10) initially thought Vancouver, Washington was too remote to be an effective test site, effective lobbying from Trent Christensen of Lake Stevens Division (the U.S. support engineer for ME Series 10) and our CAD experience with other systems convinced them it was worth their time to add us to their list.

Testing ME Series 10

A team from HP's Lake Stevens Division and Böblingen installed our first revision of ME Series 10 on our local shared network and we immediately began using it to design some of the parts for a new printer. As we went through the design process, we discovered some very impressive new capabilities, some very irritating limitations, and our quota of bugs. These observations were collected over the network and sent to Böblingen once a month via HP Desk, HP's electronic mail system. The ME Series 10 authors would then respond as they deemed appropriate.

From the outset it was apparent that we were working with a winner, although one that needed a little smoothing out. Engineers familiar with HP Draft were up to speed on ME Series 10 (creating drawings, doing isometrics, etc.) in a couple of weeks from a cold start. (The isometric shown in Fig. 1 was created by an engineer less than a week after he transferred to ME Series 10 from HP Draft.)

Yet these same users have not found the program to be

a toy that is restrictive or lacking in power. As the experience of our group using ME Series 10 progresses, we find the program a powerful tool that aids us in the development process.

One of the disadvantages of being an alpha site is the lack of manuals. It is a credit to the program that our users were able to transfer their knowledge from HP Draft to ME Series 10 so quickly using only the on-line help provided within the system augmented by startup help from the local ME Series 10 guru. Still, some of our engineers found it difficult to use the compact flowchart style information presented therein and would have preferred an illustrated, written manual.

The biggest disadvantage of being a test site for new software is the potential for losing data to software bugs. We (and several other sites) came up with several bugs, including one in the beginning that occasionally destroyed not only the drawing currently in memory, but its stored copy as well! However, if ME Series 10 had been too fraught with bugs we would have stopped using it. Because the bugs were few and quickly fixed by the authors, we used ME Series 10 throughout our development project.

Missing Capabilities

A powerful macro language is included with ME Series 10 that lets you virtually redefine the product if you desire. Still, with experience we found some language functions missing that we could not easily work around. The ME Series 10 development team was very responsive to our requests, and as each successive version came out, we were pleased to find one complaint after another disappear. The more we use the ME Series 10 macro language, the more we wish to do with it. However, it is unlikely that the authors will be able or willing to satisfy all of our requests. We noticed in our evaluation that alpha test sites never ask for fewer features, always more!

One of the limitations we found in the early versions of ME Series 10 was a lack of any layering capability. Although it provided a method of viewing parts individually (using the parts tree), no method to view groups of parts from different assemblies was provided. Layering was added and the user can now have a part on several layers at once, a nice advance over previous CAD systems.

Dimensioning was quite irritating. In trying to provide a completely automatic method for the engineer, ME Series

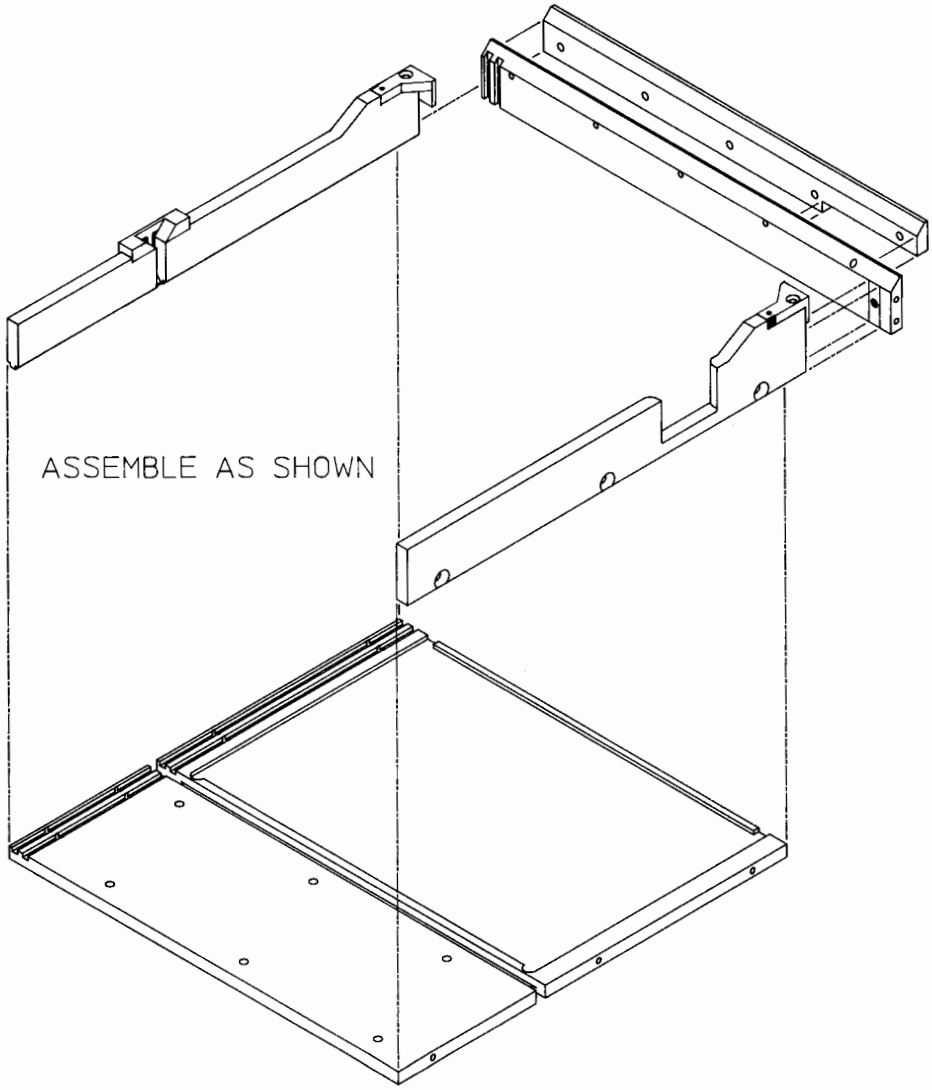

Fig. 1. *Isometric drawing created by engineer with one week's experience on ME Series 10.*

10 would place dimensions in accord with standards set up by ANSI. While admirable, not all situations are foreseen by this august body and occasionally (but too often) we found the program putting dimension text where it decided the text should be and not where we wanted it. The ME Series 10 development team saw nothing wrong with this state of affairs and would have released the product with completely automatic dimensioning intact. While we are all for ANSI standards, we did not like the program enforcing the letter of the law and convinced them that most engineers would not like it either. ME Series 10 now offers greater freedom for dimension placement.

In the early versions of ME Series 10, plotting was difficult. Although we usually prefer flexibility, when it comes to plotting we want no surprises. ME Series 10 has the capability to plot anywhere on the page and in the early versions the user was required to tell it where to plot on a page. The authors have added macros to make normal plotting (i.e., putting a D-size drawing on paper) straightforward. Users simply tell the program to send the contents of the screen to whatever paper is in the plotter. Custom positioning on a sheet is still possible for the advanced user.

When we first received the program, there were no plans for manual hatching. Automatic hatching is very fast and quite powerful but there are circumstances where the user wants to do it manually and the development team had to be convinced of this. ME Series 10 now has a manual hatching mode.

Sometimes You Don't Get What You Ask for Immediately...

There are a few functions in ME Series 10 that we feel still need development and as an alpha test site we had the opportunity to present these to the program's designers. The program's authors had several more pressing issues to attend to before the introduction of ME Series 10 and, as a consequence, some (in our opinion) needed improvements did not make it into production code. We have been told these will be incorporated in a future revision, however.

...And You Don't Always Want What You Ask for!

Sometimes the authors refused to make a requested change and we later realized that they were correct in doing so. For instance, at the beginning we requested that ME Series 10 set up the drawing area with the user-selectable "electronic sheet of paper" (A, B, C, etc.) analogy, just as HP Draft had done. We were comfortable with the analogy and felt a little lost without it. The authors refused and as we became more familiar with ME Series 10 we found that being locked into standard formats was indeed restrictive. We came to appreciate the infinite work surface of ME Series 10.

As another example, we often asked the authors to provide more macros with the base system. They politely told us to go write them ourselves. After a hesistant start we found that writing macros was as easy as they said it would be. By encouraging us to write macros, the authors showed us the means for achieving great flexibility and control. We use our fluency with the macro language to help get maximum performance from the program. We often write small procedures to help with particularly repetitive or automatable tasks (like "While I'm at lunch, plot this drawing, load that one, plot it, and then load another one.").

Some Functions Impressive from the Start

The macro language of ME Series 10 and the way it is implemented are extremely impressive and the key to very high productivity. All eight engineers using the program have developed individual menus and commands, collectively referred to as personalities. Macros are easy for the interested novice to learn, yet powerful enough for the experienced user. We have written macros ranging in complexity from simple one-line typing aids to a set that provides an on-screen, full-function RPN calculator and another set that helps an engineer design cantilever beams.

A simple screen editor is provided within ME Series 10 to help in macro development. Basic system instructions are keywords such as LINE, ARC, FIT, and EDIT_PART, which make it easy to remember command names. Because of these features, it is productive for us to create small macros as we work. If we feel some feature of ME Series 10 is awkward or a function is missing, we can usually add the function through macros or change the operation of the feature on the spot and store it for future use. This provides easy adaptability that actually gets used often.

We are continually impressed with a function designed to help construct isometric and general three-dimensional views of the part we are designing. Using AFFINE, a user can create two-dimensional isometric, dimetric, and trimetric drawings in a third of the time required to do so without

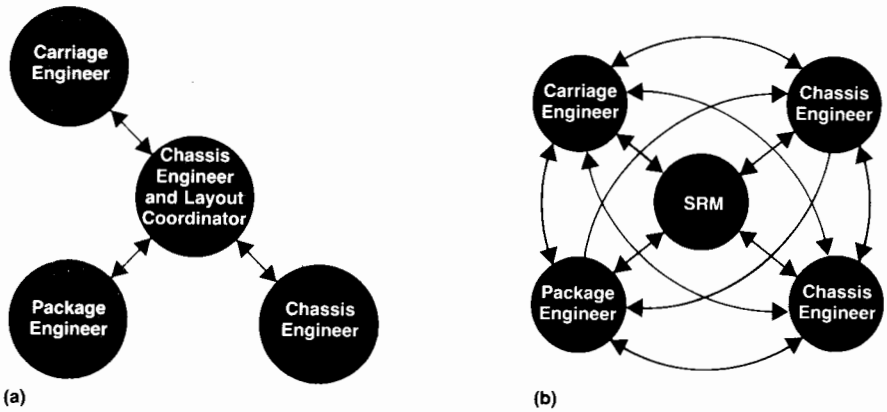

Fig. 2. *Using a group of files in an SRM directory as a development team's master layout changes the information flow within a project from (a) to (b).*

it. This functionality relieves some (but not all) of our desire for three-dimensional solids modeling, because AFFINE makes it fairly easy for us to provide the shop with general views of a part we wish to have created. As an added benefit we often find mistakes during the process of creating the isometric view, saving us time and embarrassment. With complex parts the model maker rapidly gets a good feel for what is to be created. Although the value of an isometric view is less for a simple part, the process used to create the view is then so easy that we almost always make one.

ME Series 10 supports group interaction during the design process well. The parts tree structure allows layouts to have logical part groupings. This means subassemblies can be moved around and edited easily. In fact, subassemblies can be kept in several different files and pulled in only when needed. This allows engineers to work on their designs and yet communicate changes in near real-time to the entire product team. We have our printer geometry divided between several files in a master layout subdirectory on our Shared Resource Management (SRM) network which the engineers work in and add information to. Product development time is conserved by lowering the number of build cycles required to find interferences between parts. We can do this because each designer can view on a local workstation how a particular design relates to other parts. Hence, our intragroup communications have become more effective (see Fig. 2.)

The two-dimensional analysis function built into the code was well received. It allows calculation of area, perimeter, and moments of inertia for any shape you can sketch. Dimensions can be moved around once created and automatically change in value if the part is stretched or changed. Dimensions for the whole drawing can be converted from metric units to inches with a simple toggle. Hatching stretches with the part. It is possible to change the size of a fillet after putting it in place, a great help. Three different types of layout grids are included (dot, line, and ruler). Full HP Draft construction line functionality is included. In short, ME Series 10 now has all the capability expected of a two-dimensional system plus some added special features.

Using a translator, we can move ME Series 10 drawings directly into programs for driving our model shop's NC milling machine. The existence of this link (which also exists for HP Draft) has greatly enhanced the efficiency of the shop and has yielded a noticeable drop in part defects caused by poor communication compared to the time before our link was in place.

Throughout our testing of ME Series 10 we found the development team at Böblingen to be enthusiastic and responsive to our suggestions. They were surprised that we preferred the option of having manual methods available rather than relying on automatic methods completely and learned that mechanical engineers like to do things their own way, regardless of how it is done down the street. We were impressed with the capabilities of ME Series 10 and saw an opportunity to define a tool for ourselves, getting a few steps closer to that ideal CAD station. In the process we learned a lot about the way we use our tools and in so doing increased our productivity. I would like to stress that our testing was done while we were actively designing a new printer as a team. If we had felt we were being slowed down by ME Series 10 we would have gone back to HP Draft because finishing the design and bringing our own product to market was, of course, our highest priority. The fact that we never looked back once we got up on the program is a testament to its basic functionality and the support we received from Böblingen on bug fixes.

The ME Series 10 authors now have a product that has passed through the hands of real-world users. Using us as an alpha test site involved extra effort for both groups, but each found benefits coming from the work. I hope we can do it again.

Acknowledgments

Members of the design team at VCD who tested ME Series 10 included Rick Berriman, Don Bloyer, Dave Gast, Larry Jackson, Brian King, Dave Pinkernell, and Steve Rasmussen.

Authors

May 1987

4 — CAD Workstations

Karl-Heinz Werner

Karl-Heinz Werner holds a PhD degree in mathematics from Saarbrücken University and has been at HP since 1983. He has contributed to the development of HP DesignCenter ME Series 10 and Series 30 as well as other products. Karl-Heinz was born in Völklingen and is now a resident of Deckenpfronn. He's married and has two children. His outside interests include amateur movie-making, photography, optics, and music.

Dieter Sommer

Dieter Sommer came to HP in 1984, the same year he received his degree in computer science from the University of Stuttgart. He was responsible for the tablet overlay and screen menus for HP DesignCenter ME Series 10 and is now working on product enhancements. Born in Stuttgart, Dieter lives in Magstadt and is married. He sings in a choir, plays violin, and likes dancing.

Dieter Deyke

With HP since 1979, Dieter Deyke contributed to the development of HP DesignCenter ME Series 10 and ME Series 30. He has worked on storage, a macro language, command decoding, and low-level drivers. His Diplom Ingenieur was awarded by the Engineering School at Esslingen. He was born in Göppingen and now lives in Gärtringen. He's married and has three sons and a daughter.

Wolfgang Kurz

Wolfgang Kurz joined HP in 1981 and has developed software for several CAD systems, including HP DesignCenter ME Series 10. He was born in Mühlenrahmede and studied for his Diplom Ingenieur at Paderborn. He continued his studies at the Ruhr University of Bochum and at Purdue University, from which he received an MSEE degree. Wolfgang is a resident of Böblingen and lists flying, sailing, and bicycling as his favorite recreational activities.

Heinz P. Arndt

A native of Stuttgart, Heinz Arndt studied computer science at the University of Stuttgart and received his degree in 1983. He joined HP the same year and has worked on software for various CAD systems. His design contributions on HP DesignCenter ME Series 10 include the plotter, binary storage, and parts of the internal data structure. Heinz is married and enjoys travel, sports, and reading, especially science fiction. He's also interested in personal computers and artificial intelligence.

16 — ME CAD Geometry

Karl-Heinz Werner

Author's biography appears elsewhere in this section.

Harold B. Prince

A native of Atlanta, Georgia, Hal Prince received an MA degree in music from Harvard University in 1976 and an MS degree in computer science from Yale University in 1979. He developed UNIX software for a small Los Angeles company before joining HP in 1984. His contribution for HP DesignCenter ME Series 10 centered on the parts and UNIX system interfaces. Hal lives in Gärtringen, Württemberg and enjoys reading, mathematics, and European culture.

Friedhelm M. Ottliczky

Friedhelm Ottliczky received his degree in mechanical engineering from the Technical University of Karlsruhe and joined HP in 1984. His primary contribution at HP has been the topological data structure for HP DesignCenter ME Series 10. Friedhelm was born in Künzelsau and now lives in Weil, Schönbuch. His leisure activities include classic cars, motorcycling, and skiing.

Stephen Yie

Stephen Yie joined HP's Böblingen Engineering Operation in 1983 and contributed to the development of HP DesignCenter ME Series 10. Recently, he has worked on software localization and applications and has provided technical support for ME Series 10. He has a bachelor's degree in mathematics from the University of Cologne and a 1982 doctoral degree in mechanical engineering from the Rheinisch-Westfälisch-Technische Hochschule at Aachen. Stephen was born in Jakarta, Indonesia, and now lives in Böblingen. He and his wife have three children. His recreational activities include tennis, aikido, playing piano and guitar, and Chinese cooking.

Heinz Diebel

Heinz Diebel is a native of Stuttgart and holds a Diplom Ingenieur from the University of Stuttgart. With HP since 1980, he has worked on a number of internal R&D projects and has contributed to the design of HP DesignCenter ME Series 10 and ME Series 30. Heinz now lives in Böblingen and enjoys tennis, swimming, and flying motorgliders.

30 — Alpha Site Evaluation

Paul Harmon

Paul Harmon joined HP's McMinnville Division in 1981 after receiving his BSME degree from the University of Washington. He has worked on mechanical engineering designs for a number of R&D projects and was involved in product testing for HP DesignCenter ME Series 10. His work on printhead interconnections has resulted in a patent application. In addition to his HP work, he is studying for a master's degree in mechanical engineering from Stanford University. Born in Hermiston, Oregon, Paul and his wife and new daughter live in Vancouver, Washington. He enjoys motorcycles, sports cars, programming, and church activities.

35 — Power-Line LAN

Robert A. Piety

With HP since 1972, Bob Piety is a development engineer at HP Laboratories. He's currently designing software and hardware for video graphics systems and previously characterized RF transmission properties of ac power lines, the subject of his *HP Journal* paper. Among other accomplishments, he has developed IC test systems and has designed and built digital and analog controllers. He's also working toward an MSCS degree from California State University at Chico. When Bob is not working or studying he enjoys reading, woodworking, gardening, skiing, swimming, photography, camping, electronics, and computers. Several years ago he and his wife built their current home and landscaped the grounds.