User Interaction in HP PE/SolidDesigner

The HP PE/SolidDesigner user interface is modeled after the successful, easy-to-use, easy-to-learn interface of earlier HP CAD products. All commands are coded as Common Lisp action routines. A user interface builder helps command programmers by hiding details of the X Window System and the OSF/Motif™ graphical user interface. Prototyping was done using a specially developed Lisp-based interface to OSF/Motif called HCLX.

by Berthold Hug, Gerhard J.Walz, and Markus Kühl

As the use of CAD systems has become more and more widespread, two conflicting trends have emerged. On one hand, the complexity of CAD systems has grown with their increasing functionality. On the other hand, the typical CAD system user is no longer a computer hobbyist. Designers and detailers are busy enough maintaining expertise in their own areas without having to be computer experts as well. Therefore, CAD software must be easy to learn and easy to use for first-time or occasional users without sacrificing flexibility and effectiveness for more experienced users. The conflict between the need for simple operation and the increasing functional complexity can lead not only to less user satisfaction, but also to decreased productivity. As a result, a simple and consistent user interface has been a long-standing goal of HP CAD products.

The user interface of HP PE/SolidDesigner is based on the successful user interface of HP PE/ME10 and PE/ME30. The key components of this user interface are:

- Ease of Use. The product is designed not only for experts, but also for first-time or occasional users.
- Menu Structure. A task-oriented, flat menu structure minimizes menu interaction and the length of cursor movements.
- Macro Language. This allows the user to customize the menu structure. User-defined functions can be set up to increase productivity by using existing CAD operations and measure/inquire tools for model interaction.
- Online Help System. This provides all relevant information to the user without using manuals.

The HP PE/SolidDesigner graphical user interface is based on OSF/Motif and the X Window System, universally accepted graphical user interface standards for applications software running on workstation computers. The OSF/Motif graphical user interface provides standards and tools to ensure consistency in appearance and behavior.

The large functionality built into HP PE/SolidDesigner is accessed by means of a command language with a defined syntax, referred to as *action routines*. The user communicates with the command language via the graphical user interface. All prompting, error checking, and graphical feedback are controlled by means of the command language. All CAD functionality is provided in this way, along with a user interface builder for implementing the graphical user interface.

The action routines are implemented in Common Lisp, which provides an easy and effective way of prototyping and implementing user interactions. For the first interactive prototypes, HCLX, a Lisp-based OSF/Motif interface, was used.

During the development of HP PE/SolidDesigner, HP mechanical engineers spend hundreds of days testing the product and providing feedback to tune its user interaction to meet their needs. They mercilessly complained about any awkward interactions. They made suggestions and drew pictures of how they would optimize the system for their particular tasks. As a result, commands were designed and redesigned to reflect their needs. The user interface verification was also supported by many external customer visits.

User Interface Description

If the user is familiar with other OSF/Motif-based applications, it's easy to feel comfortable with HP PE/SolidDesigner quickly. The mouse, the keyboard, and the knob box or spaceball are the tools for interaction.

When HP PE/SolidDesigner is started it looks like Fig. 1. The different areas are:

- Viewport (center of the screen). The viewport covers the main portion of the user interface and consists of the graphics area and the viewport control buttons at the top. In the graphics area of the viewport, the model is displayed and the user interacts with the model. Several viewports can exist, each with its own control buttons. Using more than one viewport, the user can view a part simultaneously from different sides and in different modes. Resizing and iconification of viewports are possible.
- Utility Area (top row). In the utility area, the user finds utility tools that support the current task. They do not terminate, but rather interrupt and support the current command. The help button at the right end gives access to the general help menu.

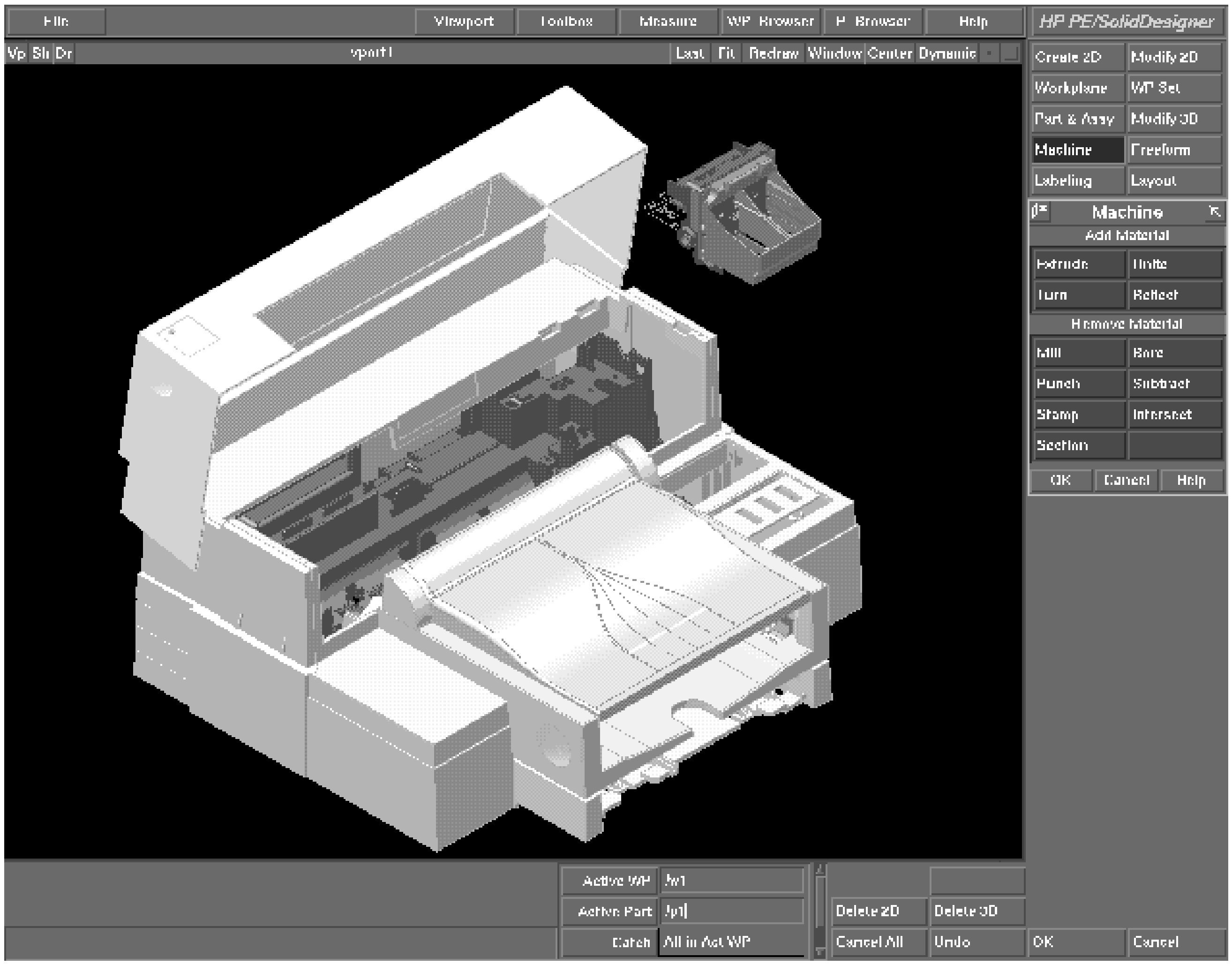


Fig. 1. Main screen of the HP PE/Solid Designer user interface.

- Main Menu (right side). The main menu buttons appear in the right column below the application name. This is also called the main task area. All the functionality is grouped into task-oriented logical areas. By selecting a main task button, the user opens a set of subtasks or a command dialog menu with buttons for all stages in the modeling sequence.
- Prompt Lines and General Entry Field (bottom left). The two-line prompt area is used for general system feedback, messages, or user guidance. The general entry field is used for entering commands, general expressions, and the like.
- Global Control Buttons (bottom right). The buttons at the bottom are always available. The select button is only active when the system is prompting the user to select something. The buttons and display fields inside the scrolled windows display general system settings like the active workplane or part, units, and catch information.*The other buttons are commands that the user needs frequently. They are always available.

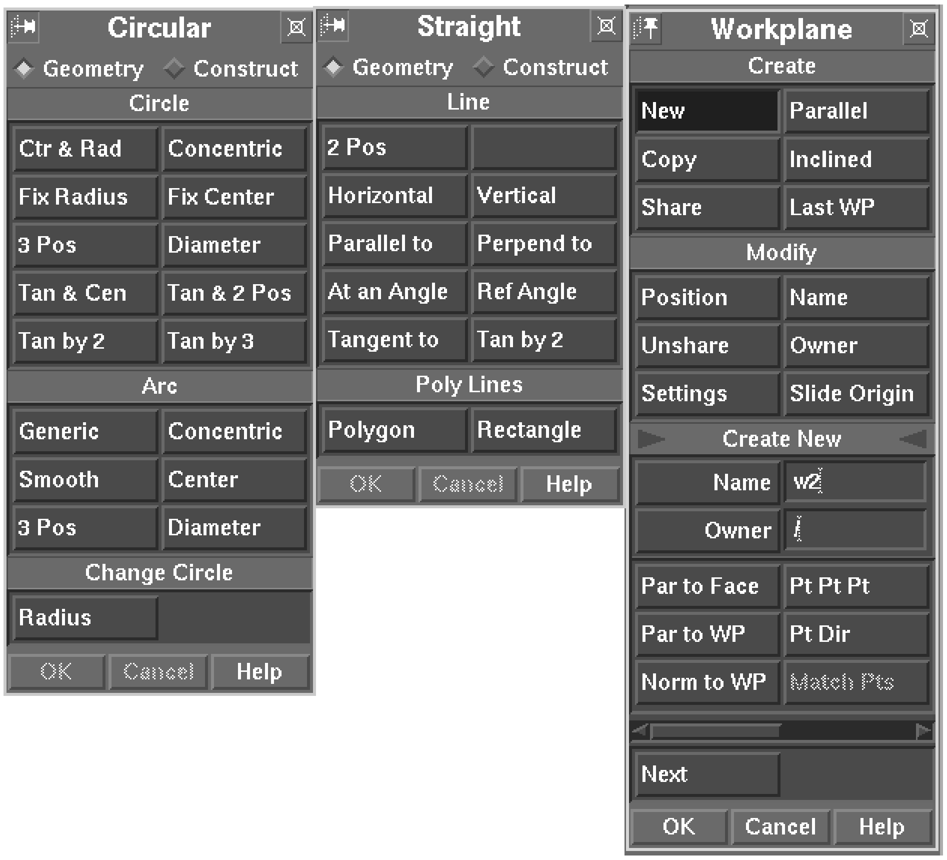
*Depending on the current command, the catch setting indicates how a pick in the graphics area (viewport) is processed to identify an element. For example, "catch vertex on current workplane" means that if the user picks near the end of a straight line, the resulting pick point will exactly match the endpoint of the line. The catch radius is customizable.

Command and Option Dialogs

Command dialog boxes (see Fig. 2) are accessed either from the main task area or the utility area. The current command dialog box is replaced by the new selected one. If the default home position of the command dialog box is inside the drawing area, the dialog box is closed upon completion of the operation (this is typical for command dialogs from the utility area). With this behavior the user always has optimal use of the screen space.

Nevertheless, sometimes the user wants to have parallel access to different dialog menus at the same time (flat structure). This can be achieved by pinning the command dialog to the screen using the small icon in the upper left corner. Pinned command dialog boxes are helpful whenever the user is using several menus constantly. The user can keep as many or as few dialog boxes open as desired and arrange them on the screen to suit the present task. Fig. 2 shows two pinned dialog boxes and one unpinned dialog box.

Activation of a command by a mouse click or by typing in a command in the general entry field leads to the same behavior. The command button snaps into pressed mode. If there exist a number of additional controls of the command, a



Name w2

Owner /

Fig. 2. Command dialog boxes with pin icons in the upper left corner. Two boxes are pinned to the screen and one is not.

subdialog is attached at the bottom of the command dialog box (see extrude box in Fig. 3). The command becomes interactive and a prompt asks for further input. The dialog box gets a yellow border, a signal that this dialog box is active. If the action is suspended by an interrupt action, the border changes to red. Thus, the user never loses track of what is active and what is not.

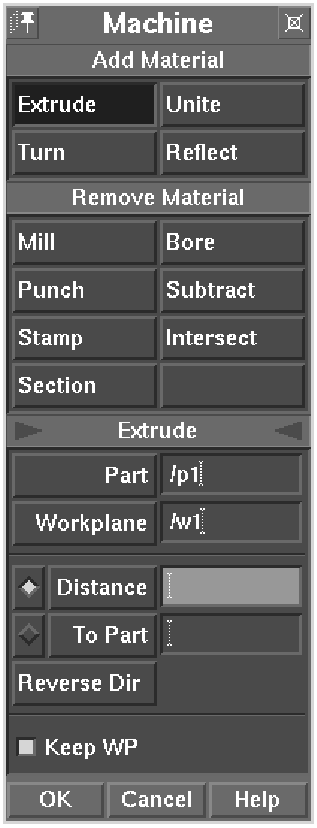
The subdialog provides options in the form of buttons, data entry fields, and check boxes for further control of the command. The system provides good defaults to minimize the required user input. All options can be manipulated in any appropriate order; the command supplies a parallel syntax. All settings are displayed in the dialog box. Required data fields are highlighted in yellow, meaning that the user must define a value.

The help buttons of the command dialog boxes give access to context-sensitive help.

Context-Sensitive Help

Help messages relating directly to the task the user is performing can be accessed immediately by pressing the help button located in the currently active menu or dialog box. The help information appears in its own dialog box, which can be positioned anywhere on the screen and resized for convenience (see Fig. 4).

Words used in help text are directly linked to other definitions or explanations. The user need not go back to indexes to look up further words to aid in understanding the help information.



Part /p1

Workplane /w1

Fig. 3. If a command has controls in addition to the basic ones, a subdialog box is attached to the command dialog box. The extrude command is an example of this behavior.

In addition to the context-sensitive help, the help system provides a task-based index with search facility, a command-based index with search facility, an overview of HP PE/SolidDesigner, information on HP PE/SolidDesigner's concepts, filters, and displays of user-typed keywords, version information, and help on help. The help system can be used in a standalone mode without running HP PE/SolidDesigner.

Task-Sensitive Tools and Feedback

Whenever the user has to enter a value for a command, the system provides the appropriate tool for data entry. For instance, if the user has to enter a direction, the direction tool (Fig. 5) pops up. The user can extract the information directly out of the model with a minimum of effort by accessing parts of the model such as edges and faces. The result is displayed either textually or graphically as part of the model.

These task-sensitive tools are implemented as subactions so that all commands (action routines) have access to the same tools. Using these tools guarantees consistent system behavior, for example in specifying directions.

Browsers

Browsers (see Fig. 6) display lists of files, workplanes, parts, and assemblies, and allow selection of items for use in commands without typing in names. Even complex assemblies become easy to understand and manipulate when browsers are used.

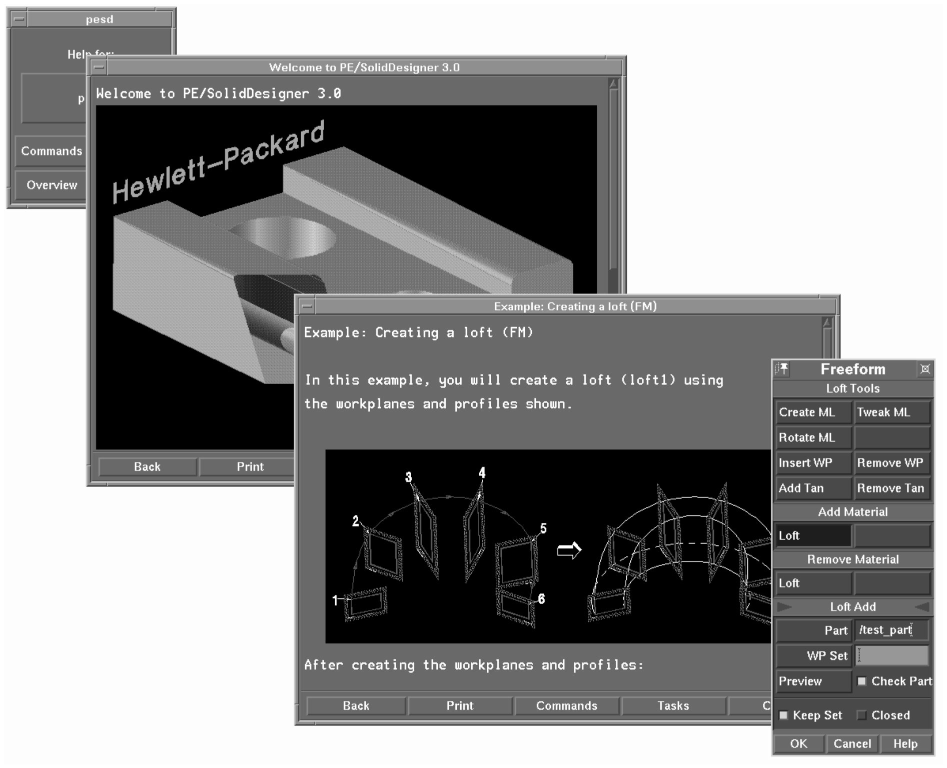
Customizing the User Interface

HP PE/SolidDesigner provides different facilities for changing its user interface. The following customization capabilities exist:

- Flattening the Menu Structure. This facility is provided by allowing the user to pin command boxes to the screen. When the environment is saved, pinning and location information is stored for later access.
- Toolbox. The toolbox (Fig. 7) allows the user to build a custom command dialog box. The user can put any command into the toolbox, and can put the most-used commands together in one area for easy access. The toolbox can be left open like a command dialog box. If a command becomes interactive, the original subdialogs are attached at the bottom of the toolbox dialog.
- Lisp. The user can write Lisp functions, which can contain action routine calls. Thus, the user can combine Lisp with CAD functionality to optimize the system for particular needs.
- Key Button Bindings. HP PE/SolidDesigner commands or Lisp functions can be accessed via X translations. Function keys, mouse buttons, or any key sequence can be defined for accessing any given functionality. This tool allows the expert user to accelerate the use of the system.
- Record/Playback. The record/playback feature allows the user to record a series of command picks to be used later to duplicate the action, like a macro. The information is stored in a file for playback. The file contains the command syntax, so it can be used to support writing user-defined Lisp functions.

Action Routines and Personality

This section describes the user interaction in HP PE/SolidDesigner in more detail. It explains the basic technology underlying the concepts that were described in the preceding section. A simplified extrude example is used to clarify the explanation.



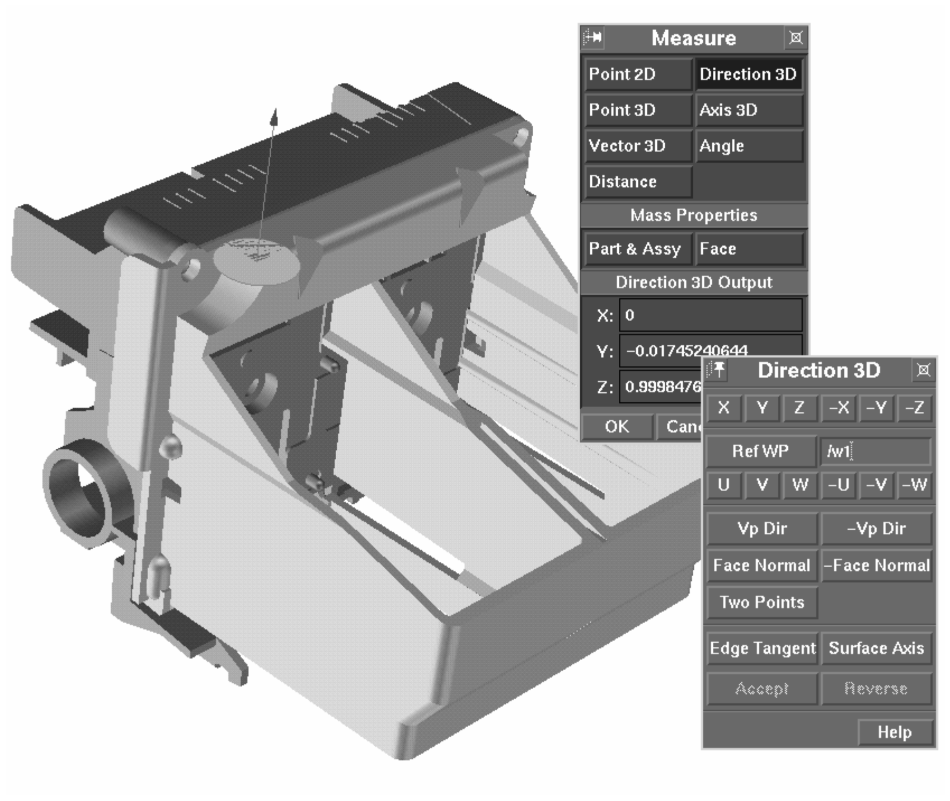
Welcome to PE/SolidDesigner 3.0

Example: Creating a loft (FM)

In this example, you will create a loft (loft1) using the workplanes and profiles shown.

After creating the workplanes and profiles:

Fig. 4. Context-sensitive help information appears in its own dialog box, which can be positioned anywhere on the screen and resized for convenience.



```
X: 0
Y: -0.01745240644
Z: 0.9998476
Ref WP /w1
```

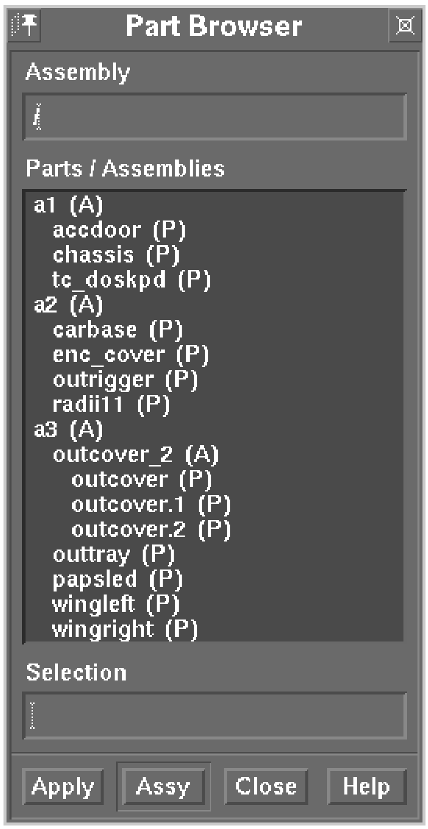
Fig. 5. When the user has to enter a value for a command, the system provides the appropriate tools for data entry. The result is displayed either textually or graphically as part of the model.

Fig. 8 is a simplified diagram of the action routine/personality communication model of HP PE/SolidDesigner. The communication model is divided into three parts. On the left side are the action routines and on the right side are the user interface objects. Bidirectional communication between the action routines and the user interface is the task of the *personality*, which is shown in the middle of Fig. 8. This division into three separate components allows the implementor of an HP PE/SolidDesigner command to change the user interface and its behavior without changing the command syntax. It is also possible to switch off the user interface for certain commands.

The action routine concept is used to implement the command language of HP PE/SolidDesigner. A command is coded as a state machine with several states and transitions between these states. The term personality refers to the information coded in the GUI update table shown in Fig. 8.

HP PE/SolidDesigner distinguishes three types of action routines:

- Terminate Actions. Terminate actions terminate every other running action routine *negatively* (i.e., they cancel them). At any time there can only be one *active* or *suspended* terminate action. All action routines that modify the solid model must be defined as terminate actions.
- Interrupt Actions. Interrupt actions interrupt the current running action routine. When the interrupt action is finished, the interrupted (suspended) action routine continues from where it was interrupted. There is no limit on the stacking of interrupt actions. Interrupt actions must not modify the solid model. They are only allowed to inquire about model data. A measure command is an example of an interrupt action.



```
a1 (A)
  accdoor (P)
  chassis (P)
  tc_doskpd (P)
a2 (A)
  carbase (P)
  enc_cover (P)
  outrigger (P)
  radii11 (P)
a3 (A)
  outcover_2 (A)
    outcover (P)
    outcover.1 (P)
    outcover.2 (P)
  outtray (P)
  papsled (P)
  wingleft (P)
  wingright (P)
```

Fig. 6. Browsers make complex assemblies easy to understand and manipulate.

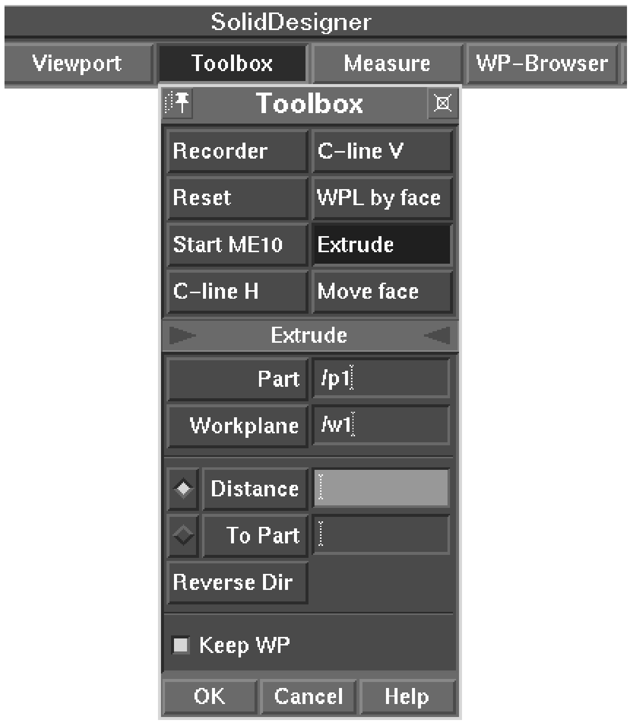


Fig. 7. The toolbox allows the user to build a custom command dialog box containing often-used commands.

- Subactions. Subactions are used to implement frequently used menus so that they can be reused in other action routines. This avoids code duplication, allows better maintenance, and improves usability. Subactions can only be called from within other action routines. This means that the user cannot call a subaction directly. Some typical examples of subactions are:
  - Select
  - Measure axis, direction, point
  - Color editor
  - Part positioning.

Basic Action Routine Structure

As mentioned above, the user interface in HP PE/SolidDesigner is Lisp-based. Therefore, the implementation of an HP PE/SolidDesigner command using the action routine concept is a kind of Lisp programming. The following is a schematic representation of a terminate action:

```
(defaction name
 ( ) ;; List of local variables (with or without initialization)
 ( ;; action description
 ( statename (state_form)
                (state_prompt)
                help-index-symbol
   ( transitionpattern  (transition_form)  pers-update-symbol  next_state
)
   ... ;; more transitions
  )
  ... ;; more states
 ) ;; end of action description
 ( ;; local functions
  (local-fun ()
   ...
 )
 ... ;; more local functions
) ;; end of local function definitions
)
```

The structure of an interrupt action or subaction is equivalent to that of the terminate action shown above except that an interrupt action is defined using the keyword defiaction and a subaction is defined using the keyword defsaction. The second parameter of the action routine definition is the

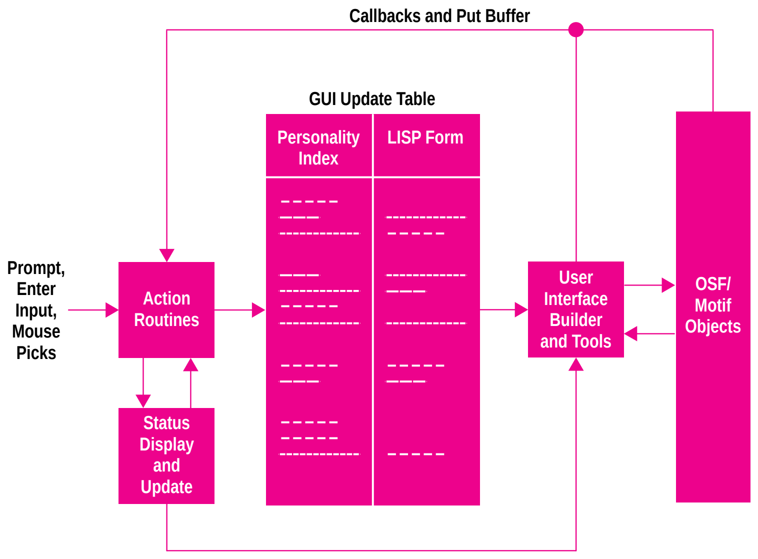


Fig. 8. In the HP PE/SolidDesigner user interface communication model, the action routines representing the commands communicate with the user interface objects through the personality.

name of the command that is coded through the action routine. For an extrude command this would be extrude. Following the command name is a list of local action variables. These variables can only be accessed from within this action routine. Action routine local functions and each state and transition form have access to them. They are used to store user-entered command parameters and as variables to control the execution of the command.

Next comes a description of the state machine. The states are those defined by the *railroad* of the command plus internal administrative states. The railroad of a command is a structure used to describe the syntax of an HP PE/SolidDesigner command for the user. Fig. 9 shows the simplified railroad of the extrude command (a few options have been omitted for clarity). The railroad reflects the concept of parallel command syntax. Each keyword (:part, :wp, :distance) can be given at any time and the command loops until the user completes or cancels it.

A distinction is made between prompting and nonprompting states. A prompting or prompt state requires the input of a token (a keyword or parameter value) from the user. This token is read from the input stream, which is filled either interactively by the user (hitting an option button, entering a number, selecting a part, etc.) or from a file (such as the recorder file). As many tokens as desired can be entered into the input buffer. Entered tokens are processed by the action routine handler. Processing stops as soon as an error occurs (such as an unknown keyword) or the input buffer becomes empty. HP PE/SolidDesigner then becomes interactive and requires more input from the user. A prompt state with an empty input buffer displays the prompt coded in its state.

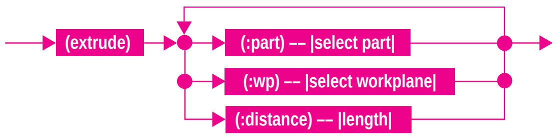


Fig. 9. Simplified railroad giving the high-level syntax of the extrude command.

After the user has entered a token, the action routine handler tries to match the input with one of the state transitions. If a match is found the action routine handler processes this transition and jumps to the next state. A nonprompting state (administrative state) takes the result of its state form to find a match with the coded transitions of this state. If the action routine handler was not able to find a match in the transitions and no “otherwise” transition was coded, it signals an invalid input error.

Implementation of the extrude railroad leads to the state machine shown in Fig. 10. As the extrude command starts, the first state is init. In this state the local variables are initialized and filled with useful defaults such as the current part and the current workplane with a valid profile. Since init is a nonprompting state and only one “otherwise” transition is coded the action routine handler goes on to the next state, top-prompt. This prompt state and the nonprompting dispatch state top-opt are the central states of this example command. The top-opt state takes the input of the previous state (top-prompt or any extract or check state) and tries to match its transitions. The states select-part and select-wp call on their only “otherwise” transition, the select subaction, as their transition form, with the specific select focus of part or

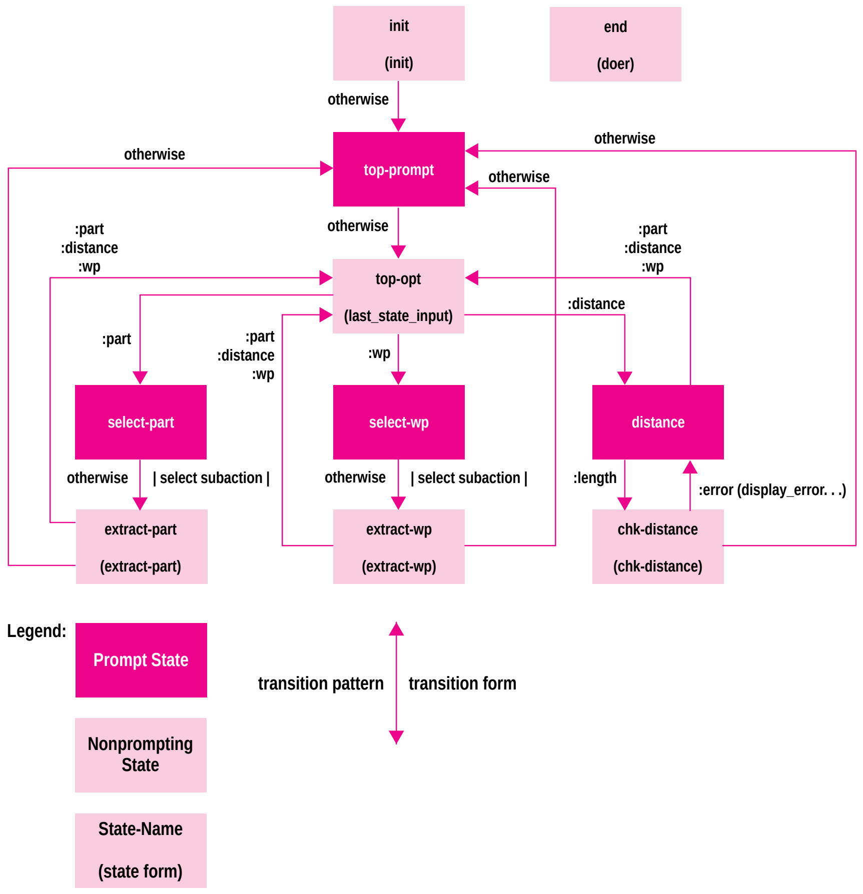


Fig. 10. State machine for the extrude command.

workplane, respectively. These states prompt through the select subaction. The extract states take the result of the transition form (select subaction call) and process the result of the select operation. The distance state has a special keyword—:length—as its transition pattern. For this keyword an input conversion is involved. The transition pattern will match any entered number, whereupon a units converter will be called automatically. A user can work in length units of millimeters or inches, and the units converter converts the length into the internal units (here mm). There are also other converters such as the angle converter which converts the user input (e.g., degrees) into internal units (here radians).

The extrude command loops until the user completes or cancels the command. In both cases the action routine handler jumps into the separated state end. Depending on a positive (complete) or negative (cancel) termination of the command, the software that actually performs the action will be called with the parameters that were collected by the action routine.

Personality

As explained earlier, the task of the personality is bidirectional communication between the action routine and the user interface objects. The core of the personality is the GUI update table shown in Fig. 8. This table stores all of the actions to be performed when an action routine executes, and it also receives data from the user. It guarantees that the user interface is in sync with the action routine state whenever HP PE/SolidDesigner requires data from the user.

The GUI update table is realized as a hash table with the pers-update-symbol (see action routine representation, page 19) as key and a Lisp form as entry. As soon as the action routine handler finds a match in the transition pattern of the current state it performs the transition form and triggers the user interface update using the third parameter of the transition definition as value. The action routine handler looks up whether a Lisp form is coded for the pers-update-symbol and evaluates it if found. The Lisp form can contain things like set-toggle of a command option or update-toggle-data to show the value the user has entered. This mechanism reflects the state of the action routine and its values at any time in the user interface.

There are special personality keywords for every action routine:

- ' action_name_ENTRY
- ' action_name_EXIT
- '(action_name action-interrupt-by-iaction)
- '(action_name action-continue-from-iaction).

' action_name_ENTRY is triggered as soon as the action routine starts. Normally the Lisp form coded for this entry ensures the display of the command options filled with all default values. ' action_name_EXIT cleans up the user interface for this command and removes the options from the screen. The other two keys are triggered when the command is interrupted or when it resumes its work after an interrupt action. In this case the coded Lisp form normally deactivates and reactivates the command options, since they are not valid for the interrupt action.

Delayed Update. A sequence of action routine calls (e.g., from the recorder file) or the input of several tokens into the input buffer should not cause constant updating of the user interface. Delayed update means that the user interface will not keep track of the action routine until the action routine becomes interactive, that is, until it requires data input from the user. At that time the user interface of the interactive command will reflect its state and values exactly.

A completely parameterized action routine does not cause any reaction on the user interface. If a command changes any status information (e.g., current part), this information will be updated. These updates bypass the GUI update table using the event mechanism.

The delayed update mechanism is implemented using a *personality entry stack*. Each trigger of a pers-update-symbol through the action routine handler will not lead to a direct execution of the Lisp form. All triggers are kept on the personality entry stack until the action routine becomes interactive. If an action routine doesn't require data from the user, all entries between and including ' action_name_ENTRY and ' action_name_EXIT are removed from the stack. As an action routine becomes interactive all Lisp forms belonging to the personality entries on the stack are performed until the stack is empty. The user interface is again in sync with the action routine state.

A problem came up with fully parameterized action routines behind a command toggle. Normally the ' action_name_EXIT trigger cleans up the command user interface, but with a fully parameterized action routine no personality trigger occurs. To solve this problem the system triggers two additional personality entries which are called in either delayed or undelayed update mode. These are ' action_name_PRE_ENTRY and ' action_name_POST_EXIT. The release of the command toggle is coded in ' action_name_POST_EXIT. The need for ' action_name_PRE_ENTRY is discussed below.

Personality Context. One requirement for the user interface of HP PE/SolidDesigner was that a command should be callable from other locations as well as from the default location. The motivation was the toolbox, which can be filled by the user with often-used commands. The main requirement was that a command's behavior in another context should be equivalent to its behavior in the default context. A user who calls the extrude command out of the toolbox expects the extrude options in the toolbox and not those in the default menu. The toolbox concept is based on the assumption that a command context is specified by:

- A calling button
- A dialog shell, in which the calling button resides
- A communication form where the command options are shown
- A shell position where the command options are shown if they are realized in a separate dialog shell.

All other things are command-specific and independent of the context.

The default context of a command is coded in ' action_name_PRE_ENTRY. Here the programmer of the command's personality defines the context in which the command should awake as the user types it in. This context can be overridden when the command is called out of, for example, the toolbox. Context dependent calls of the command personality have to check the current context settings instead of having this behavior hardcoded in the default context. This concept also makes it possible to program a totally different personality for a command or to switch off the user interface of a command.

Stacked Personality. The possibility of invoking the same interrupt action several times makes it necessary to provide a method of creating independent incarnations of the interrupt action user interface. This is done by separating the building instructions of the command option user interface into a Lisp function. As an interrupt action is called a second time (or third, etc.) after an initial invocation, the widgets of the latest command option block are renamed to save the state and contents. Then a new incarnation of the option block is created using the building instruction function. When the most recent interrupt action terminates its execution the user interface incarnation is destroyed and the widgets of the saved option block are renamed again to become valid once more. One incarnation of the option menu of a command is always kept. All other necessary incarnations are created and destroyed at run time.

User Interface Development Tools

To speed up the user interface development process a prototyping tool was required that would allow modifications to be made quickly. Since the command language of HP PE/SolidDesigner is Lisp-based and the commands are intended to interact closely with the graphical user interface (GUI), standard C/C++-based user interface builders could not be used as prototyping tools. Such tools would have required the standard edit/compile/link/test cycle, which slows down the prototyping process heavily. They also didn't offer Lisp interfaces or facilities to change the GUI of the CAD system at run time, a required feature.

In 1989 only a few Lisp interfaces to the X and OSF/Motif toolkits were available. Because none of these had all of the features we needed, we decided to produce our own. Called HCLX, it is a Common Lisp interface to the X11 Xlib, the X toolkit intrinsics, and OSF/Motif widgets (Fig. 11). It provides Lisp functions for all the functions available in libX11, libXt and libXm, as well as all the constants and resources in the X11 .h files. It provides functions to create, access, and modify all the structures used by the X toolkit and Xlib. Widget class variables are also defined, and Common Lisp functions can be used as callback routines in widgets and as functions for translations.

Although it is possible to do all X and OSF/Motif-related coding in HCLX, experience during the development process showed that certain low-level X programming should be done in C++. This includes such things as initialization, color maps, and the color button.

Color Maps. The use of a graphics library like HP StarBase and the demand for high-quality shaded solid models imply the need for a private color map within the graphics windows of HP PE/SolidDesigner. When the graphics window or its top-level shell window is focused, the graphics color map is installed (copied into the display hardware) by the X window manager. On displays that support only one color map in hardware (most of the low-end and old displays), everything on the entire screen is displayed using the installed color map. When a private color map is installed, all windows using the default color map take random colors.

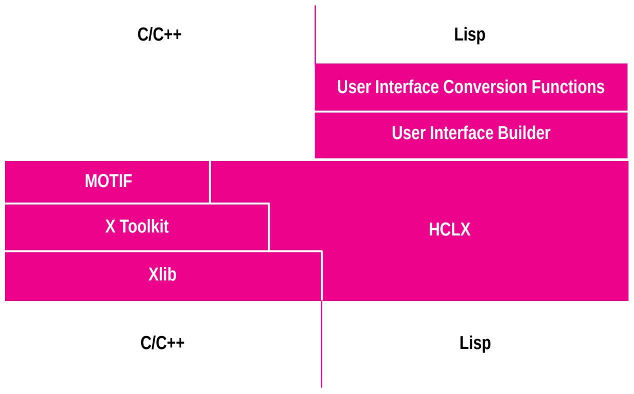


Fig. 11. Tools used to develop HP PE/SolidDesigner's user interface. HCLX is a specially created Common Lisp interface to Xlib, the X toolkit, and OSF/Motif widgets.

As soon as a window using the default color map gains the focus, the default color map is reinstalled, and the graphics windows with their private color map will have random colors. As the current color map switches back and forth from default to private, the user sees color flashing. To avoid this for the user interface of HP PE/SolidDesigner, a private color map is used for the user interface windows that has the same entries as the color map used for graphics. Along with the color map, a color converter is installed that for a given X or OSF/Motif color specification tries to find the best matching color within the color map.

Color Button. For the light settings commands, a color editor is required to give the user feedback on the colors used in the graphics windows. Therefore, a color button widget was inherited from OSF/Motif's drawn button. The color button has a small StarBase window in which colors are rendered in the same way as in the graphics windows.

User Interface Builder

HP PE/SolidDesigner's user interface builder was created using HCLX. During the prototyping phase for the user interface it became obvious that it is too expensive to train every application engineer in the basics of the X Window System and OSF/Motif. The user interface builder hides X and OSF/Motif details from the application engineer and offers facilities to create a subset of the OSF/Motif widgets.

Unique Naming. OSF/Motif widget creation procedures return a unique ID for a widget, which must be used whenever a widget is modified or referenced by some other procedure. The user interface builder changes this. Widgets are identified by unique names. These names can be specified or created automatically. The user interface builder ensures the uniqueness of the names.

Properties. For every widget only a small subset of its original resources are made available. To distinguish these resources from the full set of resources, they are called *properties*. A user interface builder property consists of a name and a corresponding value. The name is derived from the original OSF/Motif resource name by removing the prefix XmN. For example, XmNforeground becomes foreground. Some of the widget's callbacks are offered as properties. Callback properties have as a value a Lisp form, which will be evaluated when the callback is triggered. The user interface builder

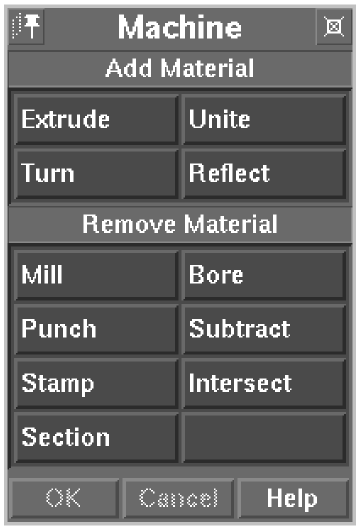


Fig. 12. Command dialog box created with a call to create-right-menu-dialog.

ensures that Lisp errors within these forms are trapped and handled gracefully. After a property has been specified for a widget, its value can be queried and the user interface builder will return the Lisp form that was used for the specification. This means that specifying red or #FF0000 as a value for the property background will result in a return of red or #FF0000 and not just a pixel value as in OSF/Motif.

User Interface Builder Action Routines. All user interface builder commands are offered as action routines. They make heavy use of the property decoders to detect input errors such as wrong property names or values. There are user interface builder commands to create widgets, modify and query widget properties, display, hide, and position widgets, and access the graphics widgets.

User Interface Convenience Functions

The user interface convenience function level is located on top of the the user interface builder level (see Fig. 11). While all the user interface builder functions are closely related to OSF/Motif, the user interface convenience functions are more abstract and not related to any window system. This level allows the programmer of a new command a fast and easy-to-use implementation of the command's user interface. The functions guarantee that the new command fits the look and feel of HP PE/SolidDesigner's user interface.

The function create-right-menu-dialog is used to create standard HP PE/SolidDesigner menus which generally appear on the right side of the user interface. The base of every right-menu dialog is a dialog shell. This allows moving and positioning these menus anywhere on the screen. A right-menu dialog can be constructed top-down with various elements. Only its width is limited to the size of two standard buttons. Fig. 12 shows a typical HP PE/SolidDesigner command dialog constructed with a call to create-right-menu-dialog.

With the function create-options-block, typical HP PE/SolidDesigner user interface objects for command options can be created. An option block can never be without a parent widget. This means that the function create-options-block doesn't

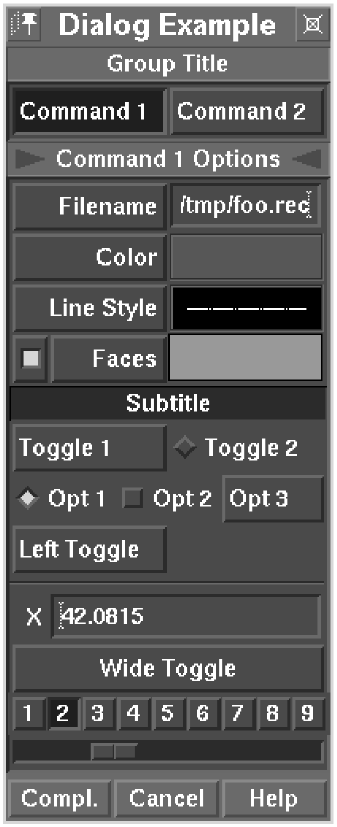


Fig. 13. Some heterogeneous option types that can be created with create-options-block.

create a dialog shell as a basis, but a form widget, which is realized in a parent widget (generally an empty form widget, also called a communication form in this article). Fig. 13 shows some of the possibilities out of which a heterogeneous option block can be constructed. Each option block has an optional title, a main part underneath the optional title, and an optional suboption form, an empty form widget below the main part as a placeholder for suboption blocks.

The function create-dialog-shell creates an empty HP PE/SolidDesigner standard dialog shell in any size. Possible elements are pin, title, close, OK, cancel, and help buttons. The empty main form can be filled with any user interface objects, which can be created using standard user interface builder calls. This function is used to create nonstandard menus such as browsers, the color editor, and so on.

Conclusion

The effort put into the development of HP PE/SolidDesigner's user interface was a good investment. The user interface is one of our key competitive differentiators. Customers like the clear structure, ease of use, and ease of learning. The Lisp-based implementation allows broad customization possibilities. The powerful concepts of HP PE/SolidDesigner's user interface and its technology provide a firm foundation for future developments.

OSF/Motif is a trademark of the Open Software Foundation in the U.S.A. and other countries.