N89-14164

1988

NASA/ASEE SUMMER FACULTY RESEARCH FELLOWSHIP PROGRAM

JOHN F. KENNEDY SPACE CENTER
UNIVERSITY OF CENTRAL FLORIDA

THE DESIGN OF AN INTELLIGENT HUMAN-COMPUTER INTERFACE FOR THE TEST, CONTROL AND MONITOR SYSTEM

Prepared By:	William D. Shoaff
Academic Rank:	Assistant Professor
University and Department:	Florida Institute of Technology Computer Science
NASA/KSC:	
Division:	Software Development
Branch:	Application Software
NASA Counterpart:	Les Rostosky
Date:	August 31, 1988
Contract No.:	University of Central Florida NASA-NGT-60002

Abstract

The graphical, intelligence and assistance capabilities of a human-computer interface for the Test, Control, and Monitor System at Kennedy Space Center are explored. The report focuses on how a particular commercial off-the-shelf graphical software package, DataViews, can be used to produce tools that build "widgets" such as menus, text panels, graphs, icons, windows, and ultimately complete interfaces for monitoring data from a application; controlling an application by providing input data to it; and testing an application by both monitoring and controlling it.

A complete set of tools for building interfaces is described in a manual for the TCMS toolkit. Simple tools create primitive widgets such as lines, rectangles and text strings. Intermediate level tools create pictographs from primitive widgets, and connect processes to either text strings or pictographs. Other tools create input objects; DataViews supports output objects directly, thus output objects are not considered. Finally, a set of utilities for executing, monitoring use, editing, and displaying the content of interfaces is included in the toolkit.

The related concepts of intelligence and assistance are explored. An HCI can be intelligent by obeying human factors guidelines; having knowledge of the application it is running; anticipating its future states; and by configuring itself to the ability of the operator. Help for the operator is provided at many levels with the knowledge possessed by the interface guiding the operator to relevant and timely information. Methods of integrating intelligence and operator assistance into the graphics module of the interface are presented.

Recommendations for how to proceed with the implementation of the TCMS toolkit are given.

Contents

1. Introduction

This report describes the design of an intelligent human-computer interface (iHCI) for the Test, Control, and Monitor System (TCMS) at NASA Kennedy Space Center. The TCMS interface will interact with hardware being developed as part of the space station project. Briefly, the interface is graphics based, uses intelligence when interacting with the operator, and provides a comprehensive help facility based on textual, graphical and audible information.

The TCMS interface is not a static object, indeed the hardware to be tested, monitored, or controlled has yet to be designed or specified. The interface built will need to be modified dynamically as hardware and applications that interact with the hardware are built and written. This lack of specification forces abandoning the idea of creating a static interface designed for one purpose, and requires, almost, the use of a magical incantation to the effect, "create an interface with a given menu structure, system message area, icons, and graphical input devices, and oh, by the way, use this knowledge about graphics and the application, and supply this help information when the operator fails to understand or needs clarification." Thus, much of this report deals with specifications for tools which can be used to invoke this incantation and thereby create a complex interface from a set of simple resources, or "widgets."

The report focuses on the graphics portion of the interface, presents requirements and a preliminary design for the TCMS interface, and discusses, less formally, features of the interface which are beyond the scope of this document, that is, the "knowledge ..., and ... help information."

The report is organized by four main concerns: monitoring output from a process, providing input to a process, creating a human-computer interface from basic parts, and using a human-computer interface. These concerns are discussed in chapters on graphics, intelligent human-computer interfaces, and help facilities for the interface.

In addition, a set of manual pages describes the facilities and tools which comprise the TCMS interface software package. These descriptions serve to define the interface by specifying the tools used to build and manipulate the interface.

Below, general requirements for the interface are given. These high-level requirements represent broad guidelines which must be following in developing the interface. More specific requirements are presented, as appropriate, throughout the report.

1.1. General Requirements

There are numerous requirements for any human-computer interface; far too many to list in this report, see [6]. However, there are certain requirements which need to be discussed.

A major requirement is that the TCMS interface obey the guidelines specified in the Space Station Information System Human-Computer Interface Guide [6]. This document provides expert knowledge on the design of human-computer interfaces. Certain of these guidelines are rules which should not be violated while others are only recommendations.

Another requirement is that the operator should be able to customize the interface to his or her own liking. For example, the user should be able to move, resize, and change the color of objects on the screen. When an operator modifies the interface, the SSIS HCIG guidelines should not be violated.

Since the hardware and application software with which the interface will interact is unspecified, the interface should be extensible, allowing new features to be added or old features to be altered as the TCMS system is developed.

The interface should be portable to many different workstations with high resolution bit-mapped graphics displays. The interface will be written in the C programming language and run under the UNIX operating system. The interface should run under the control of a windowing system such as the X window system.

The interface should provide appropriate information to the operator. This help may vary from simple panels showing the currently available options, to pages from a manual, to schematics of a system. Audible sounds should also be used to alert and inform the operator. The operator should be able to easily navigate through a complex structure of information, skipping unnecessary data and focusing only on the information selected as relevant. The interface itself

should help the operator in this traversal of information.

The interface should be constructed using commercial off-the-shelf tools. The design of this interface is quite ambitious, and clearly, starting from scratch would make it nearly impossible. If the interface is to be implemented then it is essential that appropriate interface building tools be used. Three basic tools identified in the report are: a graphics based tool to provide the widgets comprising the interface, an expert system to incorporate intelligence into the interface, and a hypertext back-end to create a multi-linked help facility containing textual and graphical information. Before a complete specification of the interface can be given each of these basic tools needs to be selected. The present discussion focuses on the graphics portion of the interface. V. I. Corporation's DataViews software package has been selected for implementing the graphics module of the interface. Some discussion of the AI tools and hypertext tools needed is given in §3 and §4, respectively.

To understand what the requirements for the TCMS might be, it is helpful to consider example scenarios for monitoring, controlling, and testing hardware. There are two fundamental problems stated in the scenarios. First, how can a interactive graphical interface be created from simple primitive objects? and second, what features should be included in the interface to make it a useful, flexible tool? These scenarios, presented in the following sections, provide an informal specification for the capabilities needed in the graphics module.

1.2. Development and Use of the Interface

This section presents, informally, four simple scenarios which could occur in the development and use of the TCMS interface. The focus of the scenarios is on the graphic capabilities of the system. To understand what the requirements for the TCMS interface might be, it is helpful to consider how one might monitor, control and test hardware, and how an operator will use the interface once it is available. The scenarios discuss creation of graphical input and output objects, merging such objects into complex objects, and the static and dynamic alteration of the objects by an operator.

1.2.1. Monitoring Applications

Consider an application programmer who has just written a program to monitor pressure at a valve. When executed, the program produces a stream of floating

point numbers directed to the standard output device. One would like to connect the output stream to a meter so that the pressure can be "seen" by an operator of the interface. In addition, one might want to store information about the valve, e.g. its name, specifications, schematic, purpose, and so on. This information can be used to provide help for the operator and to supply knowledge to the interface itself.

To create this simple interface, one needs a dynamic graphical icon representing the meter, a means of connecting the output of the valve program to the icon, and files of information about the valve. The icon, valve monitoring process, data structures connecting them, and the information files should be collected into one widget which can subsequently be included in a more complex interface.

Next, consider an application that requires input data rather than an application producing output.

1.2.2. Controlling Applications

In this scenario, an applications programmer has written a program to open or close the valve. When executed, the program synchronously read a zero or one from the standard input device which in turn closes or opens the valve. One would like to attach the input for the process to a graphic on/off toggle to operate the valve. Again certain information about the valve should be supplied as part of a help facility and a knowledge facility.

Creating a widget for this application is both similar to and different from creating the widget to monitor the valve. One still needs a dynamic icon, valve control process, data structures connecting them, and information files, but now one also needs a means of manipulating the icon to supply data to the process. This required interaction technique makes controlling an application more complex and difficult than monitoring an application. This point heavily influences the selection of tools used to build the interface.

1.2.3. Testing Applications

Next, suppose that a second application programmer has written programs which monitor and control a pump. The pump and the valve are to be joined to form a pump/valve system. Also suppose that widgets which turn on and off the pump and measure the pressure supplied by the pump have been created. One would like to merge the four widgets into one widget creating an interface

for the pump/valve system. Once the pump and valve are joined, the system can be tested turning on and off the pump and monitoring the valve pressure.

Again, information relevant to the pump/valve system should be stored and available to the interface and operator.

Other objects such as menus and message areas may be needed to complete the pump/valve interface. Thus, tools are needed to form complex structures from elemental parts. One would like these parts to be reusable, and where possible, interchangeable so that multiple customized interfaces can be constructed.

1.2.4. Operating the TCMS Interface

Finally, once the interface has been created someone, an operator, will use the interface to test, monitor, and control the hardware at NASA KSC. Inevitably, this operator will find parts of the interface unsuitable to his or her needs. For example, text sizes may be too small or too large, the default colors may be unpleasing, the functional assignment of keys may be confusing, and so on. Thus, the operator may want to customize the interface to his or her specifications. Ideally, the operator could make these changes dynamically by using a set of utilities, supplied as part of the interface, to change text fonts, colors, key mappings and so on. These dynamic changes could be saved to a start-up or *init* file which is read when the interface is initialized. Values assigned to parameters in the init file control the initial display of the interface.

The TCMS interface should monitor the user, providing log files for generating statistics about the operator and the use of the interface. It must provide help to the operator which, for the TCMS interface, may be quite varied. Finally, the interface should obey human factors guidelines and provide an efficient means for performing useful work.

2. Graphical Design of the Interface

A human-computer interface (HCI) can be rather primitive or quite complex. Indeed, such interfaces have progressed from an operator specifying circuit connections in a breadboard, to simple teletype monitors, to high resolution bit-mapped graphics workstations. A graphical interface consists of a number of objects, often called "widgets", which can be used to display data or accept data from an operator. For example, the interface may consist of menus for selecting applications to run, graphs showing the output of the applications, and message areas where system relevant information is shown. There are two related questions addressed in this chapter. First, how does one create an interface which meets the TCMS requirements as specified in §1.1, and second, how will the operator be able to effectively use the interface to do meaningful work?

A simple example showing how an interface can be constructed is provided to explain the need for tools to build an interface. The example shows how a menu can be embedded inside a window, creating a simple, but complete interface. A primary reason for proposing a toolkit for building interfaces is that the TCMS interface is not a static, well-defined object. The hardware and software with which the interface must interact has often not been specified or designed. Thus, the TCMS interface must be extensible so that new capabilities can be added as the need arises. A set of tools for creating basic graphic parts or widgets from which the TCMS interface can be constructed is proposed. Formal specifications for all facilities in the TCMS interface toolkit are given in §5.

The operator of the TCMS interface must be provided with certain utilities allowing the alteration of objects within the display. For example, the operator may wish to move or resize graphs, change the color of menus, or the fonts for textual messages. Methods by which the attributes of the interface can be controlled by the operator are are presented in §2.4 and §2.5.2.

V.I. Corporation's DataViews software package is used to build the graphic

module of the interface. Section 2.1 contains enough basic information about DataViews to make this report understandable to someone unfamiliar with DataViews.

2.1. The DataViews Software Package

DataViews, a trademark of V.I. Corporation, is comprised of two modules: DV-Draw and DV-Tools. DV-Draw is an interactive program which can be used to connect the output of a process to a dynamic graph or icon and store the process and graph as a *view* file which can later be re-played. DV-Tools is a library of graphic routines that can be used to create and manipulate views. This section is not a tutorial on DataViews and its capabilities, but does contain enough basic information about DV-Draw and DV-Tools to make the report comprehensible to someone not familiar with DataViews. The reader is referred to [1], [2], and [3] for more detailed information. DataViews will be discussed in terms of its ability to monitor output from a process, provide input to a process, facilitate the creation of user interfaces, and support user interactivity.

2.1.1. DV-Draw and DV-Tools

DV-Draw is an interactive program with which the user can select graphs from a collection of predefined bar charts, pie charts, meters, line graphs, etc., and connect these graphs to data sources, typically processes or files. Other basic primitives such as lines, rectangles, circles, polygons and text can be added to the picture created with DV-Draw. Attributes such as line style, fill style and color can be chosen for these primitives. The picture can be saved as a view file and later replayed starting the processes or opening the files and displaying the data from the processes or files using the graphs. The view file, created either using the **save** command in DV-Draw or the `Tvisave()` utility of DV-Tools, forms the basic object out of which the TCMS interface is created. The view file contains a representation of a `view` data structure shown in figure 2.1.

The drawing object contains a list of graphical objects which may be static lines, circles and rectangles, or dynamic objects such as graphs, subdrawings or color objects. DataViews supports 10 primitive graphic objects, 4 dynamic objects, and 5 non-graphical objects. The 10 primitive objects are:

1. point object (pt)

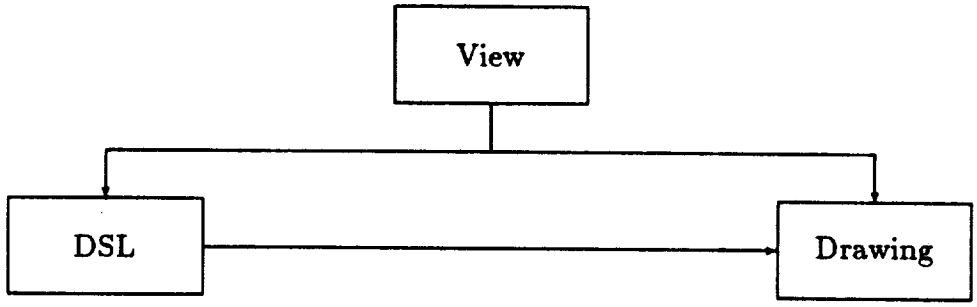

Figure 2.1: The View Data Structure

2. line object (ln)
3. circle object (ci)
4. arc object (ar)
5. rectangle object (re)
6. polygon object (py)
7. text object (tx)
8. vector text object (vt)
9. drawing object (dr)
10. subdrawing object (sd)

Variable descriptors can be associated with each attribute of the primitive objects. Altering the value of the variable descriptors and updating the display changes the appearance of the primitive.

The 4 dynamic objects are:

1. data group object (dg)
2. input object (in)
3. threshold table object (tt)

4. input technique object (it)

Data group objects present data via a graph. Input objects are graphical icons which can be used to accept data from an operator. Input techniques such as menus, valuators, and toggles are connected to the input objects. Threshold tables contain value object pairs. A variable descriptor connect to the threshold table is used to specify which object should be displayed. Threshold table are used to support dynamics.

The 5 non-graphical objects are:

1. deque object (dq)
2. color object (co)
3. transform object (xf)
4. screen object (sc)
5. location object (lo)

A deque is a data structure used to store objects. A color object is used to represent an object's foreground color. Text objects, drawing objects, data group objects, input objects and screen objects can also contain a background color. A transform object is a 3 by 3 homogeneous transform matrix that can be used to position an object. A screen object is DataViews interface to the display device. A location object represents keyboard or mouse events. It points to a key code representing the event, a screen location point and a world location point. Locator objects are used to control the interaction between the operator and the interface.

The data source list in figure 2.1 is a list of processes and files which supply data to the dynamic objects in the drawing. The data sources are connected to the objects in the drawing by means of internal variable descriptors and buffers.

It is important to notice that the flow of data is from the data source to the drawing. That is, it is not directly possible to have changes in the graphic objects serve as input for a process. Interaction handlers are required to allow input.

2.1.2. Input and Output

Any interactive human-computer interface must supply some capability for input and output. The TCMS interface requires the ability to flexibly monitor output

from processes using graphs, dynamic icons and textual displays. Input to the interface should take place from graphical devices such as menus, toggles and sliders, as well as typed text.

DV-Draw provides an easy to use method for creating graphs with attached data sources and saving these as view files. However, one can only create and save the layout or drawing for a graphical input device using DV-Draw. To attach this drawing to a process so that it can appear as an input device to a process requires writing a program using the DV-Tools library. To overcome this limitation, a collection of tools can be developed to allow an application programmer to easily connect a graphical input device to a process. These tools are discussed fully in the manual pages of §5,

DV-Tools supports 8 types of interaction handlers. There are 6 primitive interaction handlers: **VNchecklist**, **VNmenu**, **VNpalatte**, **VNslider**, **VNtext** and **VNtoggle**. Two complex interaction handlers also exist. **VNcombiner** allows a number of input objects to be combined into one input object. **VNmulti** uses a shared input area to display one of a number of input objects. The DV-Tools User's Guide and Reference Manual contain detailed information on how these interaction handlers can be used.

The interaction handlers interpret certain key presses as indicating actions to be perform. There are 5 valid action key types: `DONE_KEYS`, `CANCEL_KEYS`, `SELECT_KEYS`, `RESTORE_KEYS` and `CLEAR_KEYS`. Keyboard input and mouse events can be mapped to these keys. The TCMS interface maps `SELECT_KEYS` to one click of the left mouse button and `CANCEL_KEYS` to one click of the right mouse button. One click of the middle mouse button activates the help module of the interface. These are default key settings and can be re-mapped by the operator. When a mapped key is pressed while the cursor is in specified areas of an input object, the interaction handler returns a service result and performs defined tasks. Valid service results are `INPUT_ACCEPT`, `INPUT_DONE`, `INPUT_CANCEL`, `INPUT_USED`, and `INPUT_UNUSED`. The service result can be used to trigger other events.

The `TloPoll` utility is the DV-Tools routine is used to detect external events. There are 4 valid types of polling:

1. `LOC_POLL`
2. `PICK_POLL`
3. `WAIT_PICK`

4. WAIT_CHANGE

`TloPoll` returns a location object with key press information that can be used to control execution of the interface.

DataViews supports 40 distinct display formatters or data structures which serve as output devices, placing the actual graphic encoding of the data on the screen. In addition, custom display formatters can be written and included in DV-Tools applications or invoked from DV-Draw. Due to this rich set of graphical objects which can be used for display of data, little attention is given in this report to the creation of output displays to monitor data from a data source. Instead, it is assumed that DV-Draw will be used to create views to monitor data and custom displays will be built only when needed and then by an experienced DV-Tools programmer. Subsequent versions of the TCMS interface may include tools for building display formatter.

2.1.3. Coordinate Systems

DataViews defines a *world coordinate system* in terms of ordered pairs of integers in the range -16393 to 16383. This makes the origin the center of the drawing. User specified coordinate systems can be used when building an interface. A one-to-one aspect ratio should be used to avoid distortion of images on the display. Objects defined in a user's coordinate space are mapped by transformation objects, i.e. a 3×3 homogeneous matrices, into the DataViews world coordinate system.

An interface can be positioned on the display device using the *screen coordinate system* of the physical display on which the interface is running. This positioning is typically handled by the window manager for the display given an initial position for the interface. A *virtual coordinate system* defined by integer pairs from 0 to 32767 is used to map world coordinates to screen coordinates.

Graphic objects, in general, are not bound by position and size until the object is included inside of another object. Even so, every pickable object can be resized and repositioned dynamically.

2.2. Creating an Interface

The tools and widgets necessary to create a menu, which can be embedded in a window to make a simple interface, are discussed. Creating a menu is a nontrivial task as there are numerous design decisions which must be made, too

many to specify at a single time. For example, one must specify the selection technique, the menu style, its color, position, size, and, not least, the selectable items to be included in the menu. Some of these attributes of a menu, e.g. color, position and size, are dynamic and should not be determined until the menu is included in a window. Even then, these attributes should be alterable, within guidelines, by the operator. Other attributes, e.g. interaction technique and style, are usually defined when the menu is created. The individual menu items comprising the menu should exist before the menu is created. A bottom-up description of the interface building process is presented. This shows how individual menu items can be merged into a menu, which is then included in a window creating an interface.

2.2.1. Creating Menu Items

A menu item is a selectable graphic object together with a side effect that occurs when the object is picked. The graphic object is encoded either as a text string or a pictograph. The side effect is represented as a text string which should be the name of a process together with required or optional flags and values.

Here, a text menu is described although the TCMS interface can include iconic menus as well. To create a menu item, the interface builder must be supplied with an executable process, which, knowing its function, can be attached to a graphic object. Suppose we have a process, named `open_valve`, which operates a valve and the text string "Open Valve" which will be used as the graphic object in the menu item. When the object is selected, the process `open_valve` is activated.

First a primitive tool, called `Ctx`, can be used to create a view file containing a drawing with the text object "Open Valve." Text objects have attributes of background color, foreground color, text direction, text justification and text size together with an anchor point, specified in either world or screen coordinates, which is used to position the text. The background color is the color used for the bounding box of the text, foreground color is the color of the text, the direction can be either horizontal or vertical. The justification places the anchor point at any of 9 positions in the text (the four corners, the midpoints of the 4 edges, or the center). The size is an integer and determines the size of the text. Since text fonts are hardware dependent, one should use vector text objects for text menus. Text objects have several limitations which make them unsuitable for all but the most simple menus. For example, the background color of text objects can produce undesirable artifacts, there are only two directions in which text

can be written and the size of the text is machine dependent.

The tool `Ctx` can be invoked as shown in 2.1.

```
Ctx -t "Open Valve" -o open_value_tx.v          (2.1)
```

Here the `-t` option specifies that the string "Open Valve" is to be stored in the drawing of a view as a text object and the `-o` option redirects the output of `Ctx` into the view file `open_value_tx.v`. Options setting other attributes for the text can be used, but usually there is no need to do so until later in the interface building process. Colors, positions, and so on, should be determined when the menu item, the menu, or the interface is created.

At this step in the process, it is desirable to supply other information about the view file created by `Ctx`. For example, the SSIS HCIG requires that character heights range from 16.0 to 26.8 minutes of arc, with 20.0 minutes preferred. Using an average distance from the operator to the screen, this requirement can be converted into a height range for the text. Text sizes which are too small or too large may be disallowed. This and other human factors guidelines for text can be collected into a knowledge base of information. The knowledge base can be queried when text is altered. The TCMS interface package includes such a file, `tx.k`, which, by default, is appended to the data source list of the output view file. In addition, application specific knowledge can be appended to the view file by using the `-k` option followed by a list of knowledge base files. Finally, help files should be supplied. A default help file, `tx.hlp`, for text object is included in the output view file. Application specific help files can also be appended to the data source list of the view by using the `-h` option. At each stage in the interface building process, knowledge bases and help files can be added to the interface, see 2.5.

The next step is to attach the text view to a process and store the combination as a view file. Here the `Cmi` tool is used. It can be invoked as

```
Cmi -v open_valve_tx.v -p open_valve -o open_valve_mi.v          (2.2)
```

`Cmi` creates a view containing the text object from the file `open_valve.tx.v` and the process `open_valve`. The view is saved in the file `open_valve_mi.v`. Again, other options for `Cmi` can be selected, see §5.

The saved view file `open_valve_mi.v` can not be played as a view file and will not work as an input device. Such a menu item view file is only an intermediate widget used in the interface building process. Several menu item view files can however be collected into one menu view file.

2.2.2. Creating a Menu

Suppose now that several menu item view files have been created as in the last section. For example, suppose that menu item view files to open a valve, close a valve, start a pump and stop a pump exist. The invocation of the `Cmenu` facility, as shown in 2.3, integrates these menu items into one menu.

```
Cmenu -l open_value_mi.v close_valve_mi.v start_pump_mi.v \
stop_pump.mi.v -o pump_valve_mn.v.
```
(2.3)

Here, `Cmenu` merges the list of view files given after the `-l` option into one view and saves the result in the `pump_valve_mn.v` file.

A `-f` option can be used to have `Cmenu` read options from a file rather than from the command line. This is useful for long menu item lists.

The `-s` option allows the choice of one of 10 possible menu styles, the default being a horizontal menu bar. Other bar style menus include vertical bars, horizontal stacks and vertical stacks. Bar menus are always visible. The remaining menu styles are Pop-up, pull-down, pull-up, pull-right, pull-left and card menus. The advantage of these menu styles is that they require a minimum of display space as they are displayed only when the operator needs them, thereby reducing irrelevant information. Pop-up menu appear underneath the cursor when the appropriate keys are pressed. The pull-down, pull-up, pull-right and pull-left menus are frequently used as sub-menus of bar or pop-up menus. A card menu can be thought of as a deck of cards spread over the display. The card under the cursor is displayed and all other cards are inactive.

The `-vn` option specifies an interaction handler which defines the input technique for the menu. Currently there is only one interaction handler for menus, `VNmenu`, which is supplied as part of the DataViews software package. Using `VNmenu`, the logical names of the menu items are mapped to internal names `Item_1.text`, ..., `Item_N.text`, which are surrounded by pickable rectangles `Item_1.area`, ..., `Item_N.area`. The selection of a menu item is made by positioning the cursor inside one of the menu item areas and pressing the left mouse button. This action updates the variable descriptor associated with the menu which is used to switch between a number of choices. Depending upon the choice one of many processes is executed. See §5 for more information.

2.2.3. Creating a Window

Once a menu has been created, it can be embedded into a window. The window consists of a border, an interior region, and zero or more graphic objects. The window can be placed inside of an interface. The tool `Cwindow` is used to create a window. Options exist for selecting a border style, interior background color, position size and color for each graphic object included in the window.

In this simple example, `Cwindow` would be executed as

```
Cwindow -l pump_value_mn.v -o pump_valve_wn.v          (2.4)
```

Default borders, positions, sizes and colors are used if none are specified in the command line.

2.2.4. The Interface

The facility `Cinterface` is the highest level tool in the toolkit. An interface contains one or more windows, inside of a border. A window may be open or closed (iconic). The view file created with `Cinterface` can be played using the `play` command, described in §2.4 and §5.

To complete the example, `Cinterface` can be called as below

```
Cinterface -l pump_value_wn.v -o pump_valve_if.v       (2.5)
```

creating the interface view file `pump_valve_if.v`

Default display devices, log files, and user init files, are used if none are specified in the command line.

2.3. The Widgets Comprising the TCMS Interface

This section briefly lists all of the widgets available for creating an interface.

2.3.1. Low Level Widgets

A widget is a view file. There are 7 primitive graphic objects supported by the TCMS toolkit: arc (ar), circle (ci), line (ln), rectangle (re), polygon (py), text (tx), and vector text (vt). Each of these can be stored in a view file together with variable descriptors used to alter values for the attributes of the objects, making them dynamic. See §5.1.

2.3.2. Intermediate Level Widgets

A number of intermediate level graphical objects are useful. There are 4 intermediate level widgets in the TCMS toolkit: drawings (dr), menu items (mi), subdrawings (sd) and threshold tables (tt).

Drawings are a collection of dynamic objects and can be used to create static and dynamic pictographs and other complex graphical objects.

Menu items, as described in §2.2.2, are used as building blocks for menus. Menu items are used to create other input objects such as checklists and multiplexors.

Subdrawings are static drawings that can be included or referenced in other drawings. An include subdrawing becomes a static part of the view, which referenced subdrawings point to a view file, which if changed, changes the subdrawing as well.

Threshold tables are used to map values to objects. The value of a variable descriptor associated with the threshold table is used to select which object to display. Threshold tables are used primarily to provide color dynamics to the objects in the interface. See §5.2.

2.3.3. Input Widgets

Input objects which can be stored as views include: checklists, menus, palettes, panels, sliders, and toggles. These widgets are described in §2.1 and [2], [3]. These input objects can be grouped into compound input objects.

Future versions of the TCMS interface toolkit may also include output objects such as dials, pie charts, and so on. See §5.3.

2.3.4. High Level Widgets

High level widgets are: windows, scroll windows and icons. An icon is a closed window and it may contain an active or inactive process. Windows may be open or closed, active or inactive, and interactive or non-interactive. Scroll windows allow panning over a display.

The interface is the highest level widget. An interface can be played displaying all of the enclosed views with their processes, files and drawings.

The tools used to create the widgets described in this section are documented in the manual pages of §5.

2.4. Utilities for the Operator

An important question is how the TCMS interface will be used and modified by the operator. This section focuses on utilities for modifying the interface.

There are two methods by which the operator can control the appearance of the interface. Statically, the user can set parameters in an init file which controls the initial display of the interface, see 2.5.2. Dynamically, the user can select operations from a utility menu which is supplied automatically with every interface created using the TCMS toolkit. The utility menu contains commands to move and resize objects, select colors for objects, edit fonts, edit line styles, set action keys, request help and quit execution of the interaction session.

The TCMS toolkit also provides utilities for collecting statistics on the use of the interface, verifying the consistency of an interface against a human-computer interface knowledge base, and playing the interface. Also, there is a utility which prints a description of objects and attributes in a view file, and a utility to edit the attributes of objects in a view file. See §5.6 for more information.

2.5. The File System for the TCMS Interface

Files are used to store the views which define the interface, supply parameter values when initializing the interface, provide help information for the user, supply knowledge about the system, and record data for statistical analysis on how the interface is used. Each of these file types is discussed in turn.

2.5.1. View Files

The view files are the most important files in the TCMS interface. View files can be created either by using DV-Draw or the `TviSave()` utility of DV-Tools. A view file contains a view data structure consisting of a drawing and a data source list. Simple view files can be merged creating more complex views. Most view files can be "played" using the `play` process, which reads the view file, opens the files and processes in the data source list and displays the drawing of the screen. See §2.1 and §5 for more information.

2.5.2. The Init File

Each user can create his or her own `.tcmsrc` file which is read when an interface is executed using the `play interface` command. A default system file init

file, located in the TCMS directory `/usr/local/tcms` is used to supply default setting for the interface when the user has no `.tcmsrc` file. The format of entries in the init file consist of attribute value pairs as shown in figure 2.2.

In figure 2.2 the initial background, border colors and border thickness are set. A pop_up menu with five menu items is defined. Each menu item is then given a text name and a process to run when the item is selected. In addition, a system area with dynamic color positioned at the bottom of the screen and displaying the output of the 3 processes `time`, `ps` and `netstat`, is created.

A complete specification of all attribute value pairs that may be included in the init file is beyond the scope of this report.

2.5.3. Help Files

Help files form an important part of any human-computer interface. At each stage of the interface building process, help files may be attached to the data source list of the view being created. The TCMS interface includes default help files which can be used by the operator to show available commands, explain the use of the utility function in the Utilities menu, and provide other system information. Additional help files can be included in the interface. Typically, these additional files are used to supply assistance about a particular application. See §4 and §5 for more information.

2.5.4. Knowledge Files

Knowledge base files are used to store information on the application(s) running under control of the interface, the user of the interface, and human factors guidelines for HCI design. Default system files incorporating knowledge of human factors guidelines are supplied as part of the TCMS toolkit. Additional knowledge files can be included. See §3 and §5 for more information.

2.5.5. Log Files

A system log file (`/usr/local/tcms/adm/tcms.log`) is used to monitor use of the interface. At a minimum, the log file retains records of the user ids, time of use, and length of use of the interface. The log file can also be used to record more detailed levels of interaction. Interaction monitoring to provide a user profile is discussed more fully in §3.1.4. A user profile can be used to configure the system to a particular operator.

```
# Example init file for TCMS interface
# Line that start with # are comments

# Set attributes for the base view
WINDOW.background     blue
WINDOW.bordercolor    black
WINDOW.borderwidth    5

# Define the base menu
MENU.style            pop_up
MENU.number           5
MENU.color            red

MENU.item1.name       Utilities
MENU.item1.appl       play utilitymenu

MENU.item2.name       TCMS
MENU.item2.appl       play tcms

MENU.item3.name       Help
MENU.item3.appl       help

MENU.item4.name       Close
MENU.item4.appl       iconify

MENU.item5.name       Quit
MENU.item5.appl       exit

# Configure the system area
SYSTEM.color          dynamic
SYSTEM.number         3
SYSTEM.position       bottom

SYSTEM.item1.appl     time
SYSTEM.item2.appl     ps
SYSTEM.item3.appl     netstat
```

Figure 2.2: Example Init File

3. Intelligence in a Human-Computer Interface

The author of this report in not an expert in artificial intelligence, expert systems or other areas of cybernetics. However, the general requirement that the human-computer interface constructed for the TCMS behave in an intelligent and useful manner requires the consideration of questions, during this preliminary design stage, which deal with embedding intelligence into the TCMS interface. This chapter attempts to define intelligence in a human-computer interface and show how such an interface can be implemented.

3.1. What is an Intelligent Human-Computer Interface?

A working definition of an intelligent human-computer interface (iHCI) is

Definition 1. *An iHCI enforces good human factors guidelines, has knowledge of the application(s) running on the computer and interacting with the operator, uses this knowledge to anticipate the operator's needs and actions, and evaluates the operator's ability so as to provide a more convenient and efficient interface for the operator.*

All four features of an iHCI are discussed below.

3.1.1. Obeying Human-Factor Guidelines

Human-factor guidelines tend to be static and global. The Space Station Information System Human-Computer Interface Guide (USE-1000) specifies guidelines for the design and implementation of human-computer interfaces. The

TCMS interface is to follow these guidelines. The guide includes detailed requirements on how: (1) information should be presented to the user, (2) real-time interactions between the user and the TCMS should be handled, and (3) input of data from the user can be obtained. Each guideline is presented as a natural language rule or requirement, often with a rationale accompanying the guideline. The SISS HCIG represents a knowledge base of human factors guidelines for creating human-computer interfaces. These human factors guidelines are stable, but may change as new research in human factors engineering occurs or as technology advances. The guidelines are also global in that they do not change with different applications or operators.

It would be desirable to incorporate the SSIS HCIG into the software for the interface, so that the integrity of the interface could always be checked against the guidelines. To do so, an expert system shell capable of editing simple natural language sentences and storing these sentences as a knowledge base of rules and facts is needed. Then an inference engine that can be used to query the knowledge base whenever a change to the interface is requested.

Operators have static control over graphical and human factors attributes of the interface by using their `.tcmsrc` file, and dynamic control using the utility functions supplied with the interface, see §2.5.2 and §2.4. Whenever the interface is initiated the values set in the init file should be checked with the knowledge base for validity. Default values can be substituted for invalid settings in the `.tcmsrc` file. Similarly, whenever a utility function is used to alter the interface, the operator's use of the utility is monitored to guarantee the alteration does not violate specified guidelines. As a precaution, a tool, called `fuzz`, see §5.6 is provided to check static attributes of an interface for consistency.

There are several questions that must be settled before determining whether or not it is possible to incorporate such intelligence into the interface. These questions are discussed below, but can only be answered by people with expertise in artificial intelligence.

First, an expert system shell, which can be used to build a knowledge base of specified information together with reasoning methods about this information, must be selected from available commercial or public domain software tools. There are certain requirements that this expert system shell must meet to be acceptable for use in the TCMS interface toolkit.

The SSIS HCIG guidelines are written in natural language sentences. An expert system that recognizes true natural language would be desirable, however, the author knows of no software package capable of recognizing the guidelines

of the SSIS HCIG verbatim. Some rules may be easy to translate, while other may prove quite difficult. The power of the rule editor provided by the shell must be considered in the choice of the shell.

Assuming that the bulk of the SSIS HCIG could be incorporated as a knowledge base of information, a second question is, can the knowledge base be searched quickly enough to provide a good dialogue rate with the operator? For a system as complex as the TCMS interface, with all of its accompanying guidelines, a monolithic knowledge base would be much too rich to be useful in a highly interactive environment. To overcome such delayed interaction, the inference engine of the expert system shell should be able to access multiple independent sub-knowledge bases when asked to verify that a change of state in the interface is acceptable. For example, if an operator attempts to alter the text size of a menu item descriptor, then it should be possible to run the inference engine on a small file containing knowledge of guidelines on text and its presentation. Thus, the SSIS HCIG should be partitioned into small logically coherent knowledge bases, each small enough to be queried quickly.

Finally, since DataViews offers the ability to attach files and processes to views, the inference engine must be able to access information stored as references to files inside of view files. It is expected that any expert system shell can access knowledge base files given their names. Thus, it would seem reasonable that knowledge bases can be attached to view files as described in this report.

Partial pseudocode for the main program loop would contain code to switch between numerous choices. To allow the operator to alter attribute settings, the main control loop of the interface might be as follows.

```
decode user_action;
switch user_action
   case alter_settings:
        decode user_action;
        switch user_action
           case alter_textsize:
                check user_action against text knowledge base
                if ok, alter size
                else call help with user action and return code
                     from knowledge base check.
           case alter_color:
              .
              .
```

```
        end switch.
    .
    .
    .
end switch.
```

The pseudocode implies a primary sort on the user action to determine whether the action is to alter the setting of the interface or perform some other task. In particular, the operator may also issue commands to signal an application or to ask for assistance. The operator may even be issuing a meaningless or useless command. A secondary sort on the user action determines the specific action of the user. Before all but the most trivial action is executed, the knowledge base associated with the action can queried to determine validity of the action.

3.1.2. Knowledge of the Application

Knowledge of the application(s) attached to the interface is often dynamic and local information. For example, consider again the monitoring, control, and testing of the pump/valve system. The valve monitoring module may have access to a sub-knowledge base containing information about the valve. The valve identifier, data formats, data ranges and data thresholds sent by the module could be specified by the applications programmer when the module is written. As the valve module is integrated into the application attributes specifying how the valve is connected in the system, its backup units, its function in the system and so on needs to be given. This forms a complex network of files containing local information about parts of the interface.

When a signal is sent to the valve module the effect of the signal on the status of the interface can be determined by querying a sub-knowledge base with information about the data signal sent to the valve module. Note that this check can be performed before the signal is actually sent to operate the valve. This raises interesting implications about the power the interface possesses versus the power of the operator to control the interface. This implication is discussed further in §3.1.4.

3.1.3. Assisting the Operator

A human-computer interface "manifests its usability through the speed and accuracy with which the users can perform tasks with it; novices' ability to learn to operate the system, and sporadic users' ability to relearn to operate it; and all users' preference for operating the system" [6], [7]. Artificial intelligence can be used to aid users of the interface, making the interface faster, more accurate, and easier to learn and relearn. Help provided by the interface to the operator is discussed in more detail in §4. Ideas on how an intelligent HCI can aid the operator are discussed here. Some of these ideas are common sense rules, while others are more abstract and perhaps difficult to implement.

First, since the interface needs to respond promptly to the operator, see [6] page 3-56, the interface should anticipate the operator's actions. At each step in the interaction between the operator and the interface there is a small set of valid actions, such as, open a file, write to a file, close a file, execute a process and kill a process, with any invalid commands causing a call to the help module of the interface. The interface should anticipate possible actions by loading the appropriate files and processes from disk before they are demanded. Least recently used files and processes can be maintained in memory or fast disk. Evaluation of this pre-fetch memory management policy should be made for each port of the TCMS interface, with a "tuning" of interface memory management parameters as appropriate. Since this is an anticipatory policy, heuristics about the operator's actions, perhaps gleaned from data on the operator's previous use of the interface, may exist which can be used to decrease the system response time of the interface. Attributes of the operator may also be used to determine actions by the interface.

The TCMS interface should also assist the operator with helpful information. A request for information may be generated directly by the operator or indirectly by inappropriate actions. An intelligent HCI should be able to guide the operator through the help module in the most efficient and informative way. For example, help provided a novice may differ considerable from help offered an expert, The help offered should depend on the application, the current set of valid functions, and the actions of the operator. For example, the help offered for a pump/valve system should differ from help on a electrical circuit and direct help requests should be handled differently from error generated help requests. Heuristics for rules which determine how the help module of the interface responses need to be determined.

3.1.4. Evaluating the Operator

Should a iHCI evaluate the operator of the interface? And, if the iHCI evaluates the operator what control should be given to the operator and what control to the interface? These are both fundamental questions which must be posed and answered.

First, it is clear that the interface must monitor the operator to provide a safe, secure, and convenient system. The operator's name, login identifier, group, security level, experience level are all useful if not vital information for the interface to perform intelligently. Clearly this information must be kept secure since it can be used to allow assess for the reading and writing of files, execution of processes, and logging information about the use of the system.

The operator's experience level is a dynamic attribute that could change between sessions or during a session with the interface. Often a person's ability or experience in using an interface is evident to someone who watchs the interaction. An experienced user effortlessly moves through the interface efficiently working, while a novice will randomly strike keys or generates mouse events performing little or no useful work. By analogy, one can often tell if a musician is good or bad by simply watching how effortlessly the musician plays his or her instrument. It seems reasonable that heuristics can be developed which can be used to predict the ability of the operator and configure the interface to better suit the operator.

Several steps are required before one can propose a method for modeling the use of an interface and use the model to estimate a user's level of experience. First, do variables exist which can be used to accurately predict user ability? Can information about the operator's ability be used to configure the interface in such a manner to make use of the interface more natural to the user? Assuming the interface can be so configured, is is worthwhile (cost effective) dynamically monitor the user and alter the configuration of the interface automatically?

A search of the literature on user interface design should be conducted to determine the existence of studies identifying variables that can be used to predict user ability. If no relevant research exists, a study can be made to determine if such variables can be found. Only once such variables are identified it is reasonable to attempt to monitor these variables.

There are numerous statistics which should aid in predicting experience level. For example: total time using the interface, user interaction time, number of help requests, number of cancel selections, number of no selections, number and type of open or closed graphics in the display, the validity of the input from the

user and the responsiveness of the user to information supplied by the interface. Once a set of predictor parameters is determined, variables can be embedded as "hooks" inside of the TCMS interface to collect data on these parameters. The data can then be used to infer a level of experience.

Assume for now that the experience of the operator can be determined from variables associated with use of the interface. A basic question is whether operators of the TCMS interface should be ranked at an experience level. Some may not feel comfortable if they realized that their actions were being monitored and that this information is being used to rate them. A more positive attitude is that the monitoring is being conducted to provide a more responsive tool for the user.

There are four experience levels often mentioned in the literature: novice, intermittent, transfer and expert. An operator can be a rated by both experience in use of the interface and knowledge about the application attached to the interface. Shneiderman [7] gives many characterizations of users classified by type. Heuristics based on guidelines and studies such as these should be developed to allow configuration of the interface based on the user's profile. These heuristics would be used to control how information is presented to the user. For example, a novice user should be presented with bar style menus which are always visible, while an expert may prefer a pop-up menu. Novice users may be directed to a complete, yet succinct description of available commands, while an expert user may only need a list of key strokes and abbreviations for the commands. Inaction or inappropriate actions may be handled differently for different user experience levels.

The questions raised by the idea of incorporating intelligence into a human-computer interface are intriguing, but answers to these questions are beyond the scope of this document.

4. The Help Facility for the TCMS Interface

This chapter deals with the help facility for the interface. There are two central concerns addressed in the chapter. First, the help facility should work with an expert system to supply timely and useful information to the operator, and second the help facility should offer an intricate network of textual and graphical information through which it is nevertheless easy for the operator to navigate. Default help information on the use of the interface should always be provided and application dependent help should be easy to incorporate into the TCMS interface.

It is proposed that a hypertext system be considered for implementing the help facility. The author of this report is not an expert on hypertext, and so, only the desired properties of the help facility and methods of integrating it with the other modules of the TCMS interface are presented. These requirements and specifications serve to define the needed capabilities of the help module.

4.1. Properties of the Help Facility

The help module supplied as part of the TCMS interface should be able to display many layers of useful information to the operator. Any human-computer interface must be capable of supplying help on demand of the operator. A sophisticated HCI such as the TCMS interface requires a help facility capable of supplying help on how to use of the interface, together with information on the application(s) attached to the interface. This help information may range from simple panels of available commands, to manual pages, to schematic diagrams. Hypertext systems seem to offer this flexible network of textual and graphical information [5], [4].

An iHCI should be able to infer from the state of the interface which help information would be most useful to the operator, and if necessary display the assistance automatically. Predictive variables such as experience level, clearance level, and job code, could be used to determine how the operator is guided through the help facility. A consistent method of obtaining help should be offered. By default, one click of the middle mouse button activates the help facility.

4.2. Integrating the Help Facility into the TCMS Interface

The help module must be able to access files stored in view files. The help module must be capable of forming logical links among these stored help files and provide a convenient technique for navigating through the network of help files.

A default help system is included in the utility menu supplied with the interface. These help files serve to guide the operator in the use of the interface and are rather simple. More sophisticated help facilities need to be provided in future version of the TCMS interface.

5. Manual Pages

Manual pages for the interface are partitioned by level of complexity. Tools for creating input objects are listed in a separate section. Also system utilities to edit, play, collect statistics on, and verify the integrity of an interface are provided.

Every view file created with a tool provided by the TCMS interface toolkit can have a list of help and knowledge base files attached to the data source list of the view. The option **-a file.hlp** ... appends the listed help files to the view. The option **-k file.k** ... appends the listed knowledge base files to the view. System default help and knowledge base files are included in certain views when the view are created.

Each object has an external name by which it can be referenced. The default name of the object is the name of its type. For example, by default all objects created with `Ctx` are named `tx` and stored in the file `tx.v`. The **-o file.v** option can be used to redirect the output of a command to a named view file. The name of the file then serves as the external name of the object.

The option lists for some commands may be quite lengthy. The **-f file** option is used to force the tool to read its option list from the named file.

The example programs which appear in the DV-Tools Users' Guide provide helpful templates which can be modified to write the code for some of the tools contained in this manual.

5.1. Primitive Graphic Object Facilities

The TCMS toolkit supports 7 low level graphical primitives: arc, circle, line, polygon, rectangle, text, and vector text. These primitives can be combined to form more complex objects such as windows, menus and icons. Each of these 7 objects can be created and saved as view files using a function of the form `C??`, where `??` is a two letter abbreviation for the objects.

Each attribute of the graphical object is connected to a variable descriptor which can be used to alter the object's appearance. Often, attributes of primitive objects are set by default since the attributes more properly belong to the higher level objects created using the primitive objects. For example, when creating a menu, the textual names of the menu items may be important, and one could use `Ctx` to create these text objects. However, the position and color of the menu items need not be specified when the `Ctx` command is used.

The example program `create_view.c` in the DV-Tools User's Guide provides a template for creating the tools found in this section.

NAME

Car - create a view containing an arc object.

SYNOPSIS

Car [options]

DESCRIPTION

Car creates an arc object which is stored, by default, in the view file ar.v.
Options can be used to:
(1) append knowledge about the arc to the view file.
(2) append help documentation about the arc to the view file.
(3) redirect the output of `Car` to a view file.

OPTIONS

-c float float gives the position of the center of the arc in a user defined world coordinate system. The default center is (0.0,0.0)

-d direction specifies either CLOCKWISE or COUNTER_CLOCKWISE as the direction for drawing the arc. COUNTER_CLOCKWISE is the default.

-e float float gives the position of the end point of the arc. (-1.0, 0.0) is the default end point.

-fg color where color is one of a predefined list of foreground colors. A default foreground of white is used.

-lt line_type only SOLID_LINE is supported at present.

-lw integer specifies the width of the arc in pixels.

-s float float gives the position of the start point of the arc. (1.0, 0.0) is the default starting position of the arc.

NAME

Cci - create a view file containing a circle object.

SYNOPSIS

Cci [options]

DESCRIPTION

Cci creates a circle object which is stored, by default, in the view file ci.v. Options can be used to:
(1) append knowledge about the circle to the view file.
(2) append help documentation about the circle to the view file.
(3) redirect the output of `Cci` to a view file.

OPTIONS

-c float float gives the position of the center of the circle. A default value of (0.0, 0.0) is used.

-fg color where color is one of a predefined list of foreground colors. A default foreground of white is used.

-lt line_type only SOLID_LINE is supported at present. A default value of (0.0, 0.0) is used.

-lw integer specifying the width of the circle in pixels.

-r float float gives the position of a point on the circumference of the circle. The default is (1.0, 0.0).

NAME

Cln - create a view file containing a line object.

SYNOPSIS

Cln [options]

DESCRIPTION

Cln creates a line object which is stored, by default, in the view file ln.v. Options can be used to:

(1) append knowledge about the line to the view file.

(2) append help documentation about the line to the view file.

(3) redirect the output of `Cln` to a view file.

OPTIONS

-e float float specifies the end position in a user defined world coordinate system. A default value of (1.0, 1.0) is used.

-fg color where color is one of a predefined list of foreground colors. A default foreground of white is used.

-lt line_type only SOLID_LINE is supported at present.

-lw integer specifies the width of the line in pixels.

-s float float specifies the position in a user defined world coordinate system for the starting point of the line. A default value of (0.0, 0.0) is used.

NAME

Cre - create a view file containing a rectangle object.

SYNOPSIS

Cre [options]

DESCRIPTION

Cre creates a rectangle object which is stored, by default, in the view file re.v. Options can be used to:
(1) append knowledge about the text to the view file.
(2) append help documentation about the text to the view file.
(3) redirect the output of `Cre` to a view file.

OPTIONS

-fs fill_status determines how the rectangle is filled. **fill_status** may be one of the values FILLED_OBJECT or UNFILLED_OBJECT. A filled rectangle is drawn to its borders in the foreground color ignoring any line type and width settings for the boundary, while only the boundaries are drawn using the line type and line width attributes when the rectangle is unfilled.

-fg color where color is one of a predefined list of foreground colors. A default foreground of white is used.

-ll float float specifies the position in a user defined world coordinate system for the lower left corner of the rectangle. A default value of (0.0, 0.0) is used.

-lt line_type only SOLID_LINE is supported at present.

-lw integer specifies the width of the line in pixels.

-ur float float specifies the position in a user defined world coordinate system for the upper right corner of the rectangle. A default value of (1.0, 1.0) is used.

NAME

Ctx - create a view file containing a text object.

SYNOPSIS

Ctx [options]

DESCRIPTION

Ctx creates a text object which is stored, by default, in the view file tx.v.
Options can be used to:
(1) append knowledge about the text to the view file.
(2) append help documentation about the text to the view file.
(3) redirect the output of `Ctx` to a view file.
System default files `tx.k` and `tx.hlp` are provided.

OPTIONS

-bg color where color is one of predefined list of background colors. A default background of black is used.

-d direction specifies the text direction to be either HORIZONTAL or VERTICAL.

-fg color where color is one of a predefined list of foreground colors. A default foreground of white is used.

-j justification one of 9 possible values of logically ORing the constants AT_LEFT_EDGE, CENTERED, AT_RIGHT_EDGE with AT_TOP_EDGE, CENTERED, AT_BOTTOM_EDGE. The justification determines
how the text is placed with respect to the anchor point for the text string. A default value of CENTERED is used.

-p float float specifies the anchor position of the text in a user determined world coordinate system. The default position is (0,0).

-s size gives an integer specifying text size in hardware. A default value of 2 is used.

-t string where string is a character string which must be enclosed in double quotation marks " if the string contains white space. A NULL string is used when the -t option is not supplied.

NAME

Cvt - create a view file containing a vector text object.

SYNOPSIS

Cvt [options]

DESCRIPTION

Cvt creates a vector text object which is stored, by default, in the view file vt.v. Options can be used to:
(1) append knowledge about the vector text to the view file.
(2) append help documentation about the vector text to the view file.
(3) redirect the output of `Cvt` to a view file.
System default files `vt.hlp` and `vt.k` are provided.

OPTIONS

-d direction specifies the text direction to be either HORIZONTAL or VERTICAL.

-fg color where color is one of a predefined list of foreground colors. A default foreground of white is used.

-j justification one of 9 possible values of logically ORing the constants AT_LEFT_EDGE, CENTERED, AT_RIGHT_EDGE with AT_TOP_EDGE, CENTERED, AT_BOTTOM_EDGE. The justification determines
how the text is placed with respect to the anchor point for the text string. A default value of CENTERED is used.

-p float float a pair of numbers specifying the anchor position of the text in a user determined world coordinate system. The default position is (0,0).

-t string where string is a character string which must be enclosed in double quotation marks " if the string contains white space. A NULL string is used when the -t option is not supplied.

-ta float a number giving the angle in degrees from the text base line to the horizontal. A default value of 0.0 is used.

-tc float a number giving the intercharacter spacing of the text. Normally set to 0.0, the character spacing represents a fraction of the character width to add between characters.

-tf font one of a set of 15 Hershey fonts can be specified.

-th float sets the text height with respect to the text baseline. Normally set to 1.0 giving a default size of 1024 default world coordinate unit high.

-tl float a number giving the interline spacing of the text. Normally set to 0.0, the line spacing represents a fraction of the character height to add between lines.

-ts float sets the text slant by specifying the angle in degrees by which the text is rotated from normal toward the rotated text baseline.

-tw float sets the text height with respect to the text baseline. Normally set to 1.0 giving a default size of 1024 default world coordinate units high.

5.2. Intermediate Level Facilities

This section of the manual presents the tools: `Cdr`, `Cmi`, `Csd` and `Ctt`. which are used to create drawings, menu items, subdrawings, and threshold tables. The primitive graphical objects which can be created using the low level facilities of section 5.1 are not complex enough to build a complete human-computer interface. It is convenient to define an intermediate level of tools which can be used to create intermediate widgets such as menu items and pictographs that can be later included in higher level objects.

Pictographs can be constructed using the low level graphical objects created using the utilities described in §5.1. These pictographs are created using the facilities `Cdr` and `Csd`, which create drawing and subdrawing objects. The `Cmi` facility is used to connect a process to a text, vector text string, or a pictograph object. The process can be signaled when the object is selected. The `Ctt` facility can be used to create a collection of graphic objects stored in a threshold table. A look-up valve is used to select which of the objects in the table to display.

The program `view_merge.c` from the DV-Tools User's Guide provides a template for `Cdr` and `Csd`. Creation of threshold tables is shown in `view_create.c`

NAME

Cdr - create a view file containing a drawing object.

SYNOPSIS

Cdr [options]

DESCRIPTION

Cdr creates a drawing object which is stored, by default, in the view file dr.v. Options can be used to:
(1) append knowledge about the drawing to the view file.
(2) append help documentation about the drawing to the view file.
(3) redirect the output of `Cdr` to a view file.
Drawing objects contain a deque of objects and a symbol table of names for every named object in the deque. `Cdr` is used to merge a number of separate graphical primitives into more complex drawings.

OPTIONS

-bg color where color is one of a predefined list of background colors. A default background of black is used. The flag NO_BACKGROUND means that the background is to be transparent.

-fg color where color is one of a predefined list of foreground colors. A default foreground of white is used.

-l file1.v name1 ... specifies a list of (view file, name) pairs. The view file contains the graphical object to be merged into the drawing and the name is a text string used to identify the object. If the name is missing, the default name of the object stored in the view file is used.

NAME

Cmi - create a selectable graphic object with an associated side-effect.

SYNOPSIS

Cmi [options] -p process -v file.v

DESCRIPTION

Cmi combines either a text object, vector text object, or an icon specified by the -v options with a process to form one view file that can be included as part of a menu. The process specified by the command line is a text string, which should be enclosed in double quotes " if it contains white space. The process is a process name together with an optional list of arguments. The process name is stored in the data source list of the view and the graphic object is stored in the drawing for the view. The argument list for the process, if present, is stored as a text object in the drawing. A border for the menu item is drawn as a bounding box for the graphic object. Note that the position, size, interaction technique and highlighting method are not specified when a menu item is created since these are properties of the menu and the interface.

OPTIONS

-b border specifies the style of the border. Values for border are BOX, NESTED_BOX, and NO_BORDER.

SEE ALSO

Cmenu(3)

NAME

Csd - create a view file containing a subdrawing object.

SYNOPSIS

Csd [options]

DESCRIPTION

Csd creates a drawing object which is stored, by default, in the view file sd.v. Options can be used to:

(1) append knowledge about the subdrawing to the view file.

(2) append help documentation about the subdrawing to the view file.

(3) redirect the output of **Csd** to a view file.

Subrawing objects are static drawings with no dynamic elements. When subdrawings are saved in a view file, they are included or referenced. Included subdrawings are copied into the view file, while referenced subdrawings store only the file name of the drawing view file.

OPTIONS

-fg color where color is one of a predefined list of foreground colors. A default foreground of white is used.

-p float float specifies a center point for the subdrawing. A default value of (0,0) is used.

NAME

Ctt - create a view file containing a threshold table object.

SYNOPSIS

Ctt [options]

DESCRIPTION

Ctt creates a drawing object which is stored, by default, in the view file tt.v. Options can be used to:
(1) append knowledge about the threshold table to the view file.
(2) append help documentation about the subdrawing to the view file.
(3) redirect the output of `Csd` to a view file.
A threshold table object is used to map a value to an object. The value of a variable descriptor determines which object from the table will be displayed.

OPTIONS

-rgb value integer integer integer ... defines a red-green-blue color threshold table. The integers should be in the range 0 to 255.

-lu value index... defines a color look-up threshold table that used the index to access the display device's color look-up table.

-l value file.v ... specifies a threshold table contain the objects found in the view files of the list.

5.3. Input Facilities

There are 8 tools which can be used to create input devices: Cchecklist, Ccombiner, Cmenu, Cmulti, Cpalette, Cpanel, Cslider and Ctoggle. See §2.1 and 2.1.2. Each input device is connected to an interaction handler specified by the `-vn` option with the interactions handlers supplied by DataViews software used as defaults. A layout for the interaction handlers specified by the `-v` option determines how the input object will appear. The layout file must conform to the specification for labeled areas and text strings. See the DV-Tools Users' Guide and Reference manual for details. Select, cancel, done, restore and clear keys can be set using the **`-select string`**, **`-cancel string`**, **`-done string`**, **`-restore string`** and **`-clear string`** options. The string supplied to these options should be enclosed in " if it contains white space.

The programs `IH_menu.c` and **`forms`**`.c` provide templates for creating input technique view files.

NAME

Cchecklist - create a checklist of objects and associated actions.

SYNOPSIS

Cchecklist [options] -l item1.v ... itemN.v

DESCRIPTION

A checklist allows selection from a list of objects. The list of objects are pictographs with associated check areas. A check symbol is displayed in the check area when the object is selected. Each object has a corresponding variable which is set to 1.0 or 0.0 when the object is selected or deselected. The list of view files specified in the command line should have been created using the `Cmi` tools. When the input from the checklist is accepted by the operator, the processes associated with each checked item is executed.

OPTIONS

-bg color specifies the background color. Black is the default.

-c file.v is used to define a check symbol. A default symbol is used if this option is not specified.

-fg color specifies the foreground color. White is the default.

-m integer integer specifies layout for objects as a matrix of rows and columns.

-p float float specifies the position of the checklist.

-vn interaction_handler Only VNchecklist is provided as an interaction handler.

BUGS

More than one checklist interaction handler should be available.

NAME

Ccombiner - create a combination of input objects.

SYNOPSIS

Ccombiner [options]

DESCRIPTION

A combiner interaction handler allows a collection of checklists, menus, palettes, panels, sliders and toggles to be combined into one input object. At least one input object should be specified in the command line option list. Placement of each input object included in the combination is specified by a pair of coordinates giving diametrically opposite corners for the input object. The coordinate system may be chosen by the user.

OPTIONS

-bg color specifies the background color. Black is the default.

-fg color specifies the foreground color. White is the default.

-c file1.v (x_1, y_1), (x_2, y_2)... specifies a list of checklists objects together with the positions of diametrically opposite corners for each checklist.

-m file1.v (x_1, y_1), (x_2, y_2)... specifies a list of menus objects together with their positions.

-pn file1.v (x_1, y_1), (x_2, y_2)... specifies a list of panel objects together with their positions.

-pl file1.v (x_1, y_1), (x_2, y_2)... specifies a list of palette objects together with their positions.

-s file1.v (x_1, y_1), (x_2, y_2)... specifies a list of slider objects together with their positions.

-t file1.v (x_1, y_1), (x_2, y_2)... specifies a list of toggle objects together with their positions.

-vn interaction_handler Only VNcombiner is provided as an interaction handler.

BUGS

More than one combiner interaction handler should be available.

NAME

Cmenu - create a menu by combining one or more menu items.

SYNOPSIS

Cmenu [options] -l item1.v ... itemN.v

DESCRIPTION

Cmenu creates a menu from a number of existent menu items which have been stored as view files. The menu is stored as a view file. The menu style, trigger for selection, and highlighting method are set by default to BAR, click left mouse button inside item area, and toggle border between thick and thin when the cursor is within the menu items.

OPTIONS

-bg color specifies the background color. Black is the default.

-fg color specifies the foreground color. White is the default.

-e echo defines how menu items will be echoed. Valid values are BORDER, which is the default, FILL, and NONE. The BORDER option toggles the line thickness of a text menu item between thick and thin. The bounding box of a icon menu item is drawn when the BORDER option is used. FILL toggles the fill of the menu item area between filled and unfilled and is valid only for text menus. The menu is never highlighted when the NONE option is specified.

-p poll controls whether or not the menu item highlights whenever the cursor is inside of the menu item. The default is YES which highlights the menu item whenever the cursor is inside of the menu item bounding box. NO highlights the item only when the pick or selection is made.

-s style_flag There are 10 available menu styles: H_BAR, V_BAR, H_STACK, V_STACK, POP_UP, PULL_DOWN, PULL_UP, PULL_LEFT, PULL_RIGHT, and CARDS.

-sp space_flag Setting the space flag to NO causes the last highlighted menu item to remain highlighted when the cursor is not in the menu item area. The default setting is NO.

-st status_flag setting the status flag to YES causes the menu's control variable to be used to highlight the corresponding menu item when the menu is initially drawn. The value NO highlights no initial menu item.

-vn interaction_handler Only VNmenu is provided as an interaction handler.

BUGS

More than one menu interaction handler should be available.

NAME

Cmulti - create a multiplexor allowing one input technique to be shared by several input objects.

SYNOPSIS

Cmulti [options] -m file.v -l file1.v ...

DESCRIPTION

A menu, specified by the -m option, is used to select which of the input objects listed after the -l option

OPTIONS

-bg color specifies the background color. Black is the default.

-fg color specifies the foreground color. White is the default.

-e echo defines how menu items will be echoed. Valid values are BORDER, which is the default, FILL, and NONE. The BORDER option toggles the line thickness of a text menu item between thick and thin. The bounding box of a icon menu item is drawn when the BORDER option is used. FILL toggles the fill of the menu item area between filled and unfilled and is valid only for text menus. The menu is never highlighted when the NONE option is specified.

-p poll controls whether or not the menu item highlights whenever the cursor is inside of the menu item. The default is YES which highlights the menu item whenever the cursor is inside of the menu item bounding box. NO highlights the item only when the pick or selection is made.

-s style_flag There are 10 available menu styles: H_BAR, V_BAR, H_STACK, V_STACK, POP_UP, PULL_DOWN, PULL_UP, PULL_LEFT, PULL_RIGHT, and CARDS.

-sp space_flag setting the space flag to NO causes the last highlighted menu item to remain highlighted when the cursor is not in the menu item area. The default setting is NO.

-st status_flag setting the status flag to YES causes the menu's control variable to be used to highlight the corresponding menu item when

the menu is initially drawn. The value NO highlights no initial menu item.

-vn interaction_handler Only VNmulti is provided as an interaction handler.

BUGS

More than one menu interaction handler should be available.

NAME

Cpalette - create a color palette.

SYNOPSIS

Cpalette [options] -v file.v

DESCRIPTION

Cpalette creates a palette from a threshold table. The view file specified by the -v options should be created with the `Ctt` tool.

OPTIONS

-bg color specifies the background color. Black is the default.

-fg color specifies the foreground color. White is the default.

-p poll controls whether or not the palette highlights whenever the cursor is inside of the palette. The default is YES which echos the palette selection whenever the cursor is inside of a palette item. NO highlights the item only when the pick or selection is made.

-vn interaction_handler Only VNpalette is provided as an interaction handler.

BUGS

More than one palette interaction handler should be available.

NAME

Cpanel - create an text input panel for a single line of text.

SYNOPSIS

Cpanel [options] -p process -v file.v

DESCRIPTION

Cpanel creates a text input panel. A view file file.v is used to supply a drawing for the panel. The view file must contain a closed region for input labeled Text_Input.area and optionally four regions labeled Restore.area, Clear.area, Done.area, and Cancel.area. The text input area defines where the input string will be displayed. If the string is too long to fit within the text input area, the string is scrolled to the left. The restore, clear, done, and cancel areas define buttons which respectively restore the input string to its original value, set the input string to NULL, signal completion of the input, and abort the interaction returning a cancel code and NULL string.

OPTIONS

-bg color specifies the background color. Black is the default.

-fg color specifies the foreground color. White is the default.

-b bell_flag A value of YES (default) sounds the bell when there is too much text for the interaction handler to accept.

-c caret_flag uses the caret symbol ^ to mark the current cursor location. Values are YES (default) and NO.

-vn interaction_handler Only VNtext is provided as an interaction handler.

BUGS

More than one text interaction handler should be available.

NAME

Cslider - create a slider (valuator) for input of a floating point number.

SYNOPSIS

Cslider [options] -p process -v file.v

DESCRIPTION

Create a slider input object for floating point values. When the slider value is accepted, the value is sent as input to the process.

OPTIONS

-bg color specifies the background color. Black is the default.

-fg color specifies the foreground color. White is the default.

-dn file.v float float specifies a DOWN button view file and its position.

-inc float controls the percentage of the variable range by which the slider position changes when the UP and DOWN buttons are selected.

-p poll controls whether or not the slider highlights whenever the cursor is inside of the slider. The default is YES which echos the slider selection whenever the cursor is inside of a slider item. NO highlights the item only when the pick or selection is made.

-r min max specifies a range for the slider. The default is 0.0 to 1.0.

-t type specifies either a slider or scrollbar representation when drawing the slider. Valid types are SLIDER and SCROLL.

-up file.v float float specifies a UP button view file and its position.

-var string float float controls position and appearance of a variable string describing the slider data.

-vn interaction_handler Only VNslider is provided as an interaction handler.

BUGS

More than one slider interaction handler should be available.

NAME

Ctoggle - create a toggle which can send discrete values to a process.

SYNOPSIS

Ctoggle [options] -p process -v file.v

DESCRIPTION

Ctoggle creates graphic switch that is attached to the input pipe of a process. The view file contains either a sequence of text strings or graphic icons. If a list of text strings is specified, the display will toggle through the list. If text strings are not used the sequence of iconic objects stored in the drawing of the view file will be sequentially displayed. By default the display list wraps around cyclicly.

The processes associated with the selected option will execute when the input is accepted.

OPTIONS

-bg color specifies the background color. Black is the default.

-fg color specifies the foreground color. White is the default.

-vn interaction_handler Only VNtoggle is provided as an interaction handler.

-w wrap_flag controls whether the display cycles from the last to the first object in the object list or bounces between the last object and first object through intermediate objects. The default is YES, causing the display to cycle, while a value of NO causes the display to bounce.

BUGS

More than one toggle interaction handler should be available.

5.4. Output Facilities

At present, the TCMS interface package does not contain tools for creating output monitoring widgets such as line graphs, bar charts and pie charts. DV-Draw provides a convenient, interactive environment for creating such objects. Later versions of the TCMS interface package can provide facilities for creating these widgets. This would promote a consistent style for creating the interface, it may be more efficient, and it may provide for more power in creating output monitoring widgets. See Display Formatters [2] and Writing Display Formatters [3].

5.5. High Level Tools

The high level facilities: Cicon, Cinterface, Cscroll and Cwindow are presented in this section of the manual.

An icon is a closed window. A window can be either open or closed, active or inactive. An open active window can either be interactive or noninteractive. A scroll window is an interactive window which can be scrolled horizontally, vertically or both.

The tool `Cinterface` is used to create an interface which can be played.

The specification of options for the facilities contained in this section are not complete.

NAME

Cicon - create an icon.

SYNOPSIS

Cicon [options] -cl file.v (x,y) -op file.v (x,y)

DESCRIPTION

Cicon is used to create an icon. An icon is a closed window. The -cl option specifies the position an drawing for the closed icon. The drawing should have been created using the `Cdr` tool. The -op option specifies the position and drawing for the open window. The open window should have been created using the `Cwindow` tool. Selecting a closed icon opens it. An open window can be iconified using the "Utilities" menu provided with the interface.

OPTIONS

-fg color where color is one of a predefined list of foreground colors. A default foreground of white is used.

BUGS

The specification of `Cicon` is incomplete.

NAME

Cinterface - create a interface from a collection of view files.

SYNOPSIS

Cinterface [options]

DESCRIPTION

Cinterface is used to create an interface. An interface must contain one or more menus. Each interface contains a utility menu which can be used by the operator to change the attributes of the graphic objects within the interface. The utility menu is supplied as part of the TCMS interface toolkit. Application menus can optionally be added to the interface.

OPTIONS

-bg color where color is one of a predefined list of background colors. A default background of black is used.

-i file.v $(x_1, y_1)(x_2, y_2)\ldots$ specifies a list of icons and their positions.

-m file.v $(x_1, y_1)(x_2, y_2)\ldots$ specifies a list of menus and their positions.

-s file.v $(x_1, y_1)(x_2, y_2)\ldots$ specifies a list of scroll windows and their positions.

-w file.v $(x_1, y_1)(x_2, y_2)\ldots$ specifies a list of windows and their positions.

BUGS

The specification of `Cinterface` is incomplete.

NAME

Cscroll - create a scroll window.

SYNOPSIS

Cscroll [options]

DESCRIPTION

Cscroll is used to create a scroll window.

OPTIONS

-bg color where color is one of a predefined list of background colors. A default background of black is used.

BUGS

The specification of `Cscroll` is incomplete.

NAME

Cwindow - create a window contain other graphic objects.

SYNOPSIS

Cwindow [options]

DESCRIPTION

Cwindow is used to create a complex graphic object consisting of one or more simple objects.

OPTIONS

-b border specifies the style of the border. Values for border are BOX, NESTED_BOX, and NO_BORDER.

-bg color where color is one of a predefined list of background colors. A default background of black is used.

-l file.v float float ... list of objects and positions for inclusion in the window.

BUGS

The specification of `Cwindow` is incomplete.

5.6. Utility Tools

There are several utilities which support the interface.

NAME

colstat - collect statistics of the use of the interface.

SYNOPSIS

colstat [options]

DESCRIPTION

Statistics on the use of the interface are collected from a log file of statistics. The system default file (`/usr/local/tcms/adm/tcms.log`) is used as the log file, unless another file is specified with the -f option.

OPTIONS

-c sort output by command.

-d sort output by date.

-f file use the specified file for statistics.

-s sort output by user time on the system.

-u user print statistics for the specified user.

BUGS

Specification of which user interactions should be recorded in the log file has not been determined.

NAME

edview - edit a view file.

SYNOPSIS

edview [options] file.v

DESCRIPTION

Edview is used to change attributes, control points, and names of objects contained inside of a view file. Each primitive of object supported by the TCMS interface can be edited by supplying an attribute-value list. The attribute indicates which characteristic of the object to change and the value gives the characteristics new value. Compound objects can be edited by specifying the name of a primitive object contained in the compound object together with an attribute value pair.

OPTIONS

-ar attribute-value list

-ci attribute-value list

-ln attribute-value list

-re attribute-value list

-py attribute-value list

-tx attribute-value list

-vt attribute-value list

-dr name-attribute-value list

-mi name-attribute-value list

-sd name-attribute-value list

-tt name-attribute-value list

-checklist name-attribute-value list

-menu name-attribute-value list

-palette name-attribute-value list

-panel name-attribute-value list

-slider name-attribute-value list

-toggle name-attribute-value list

-icon name-attribute-value list

-scroll name-attribute-value list

-window name-attribute-value list

-interface name-attribute-value list

SEE ALSO

prview(6)

NAME

fuzz - verify setting in the interface against a knowledge base of human-computer interface guidelines.

SYNOPSIS

fuzz [options]

DESCRIPTION

Fuzz is similar to the lint routine which detects features of a C program that are likely to be bugs, non-portable, or wasteful. Fuzz is used to verify that an interface does not violate human-factor guidelines.

BUGS

There are some guidelines which can not be checked by fuzz, because they deal with the dynamic display of the interface.

NAME

play - play an interface, reading its view file opening the processes and files in its data source list and displaying its drawing.

SYNOPSIS

play interface

DESCRIPTION

Play is use to execute an interface. The code given in `playback.c` from the DV-Tools User's Guide provides a template for the play facility.

BUGS

Only view files created with the Cinterface command can be played.

NAME

prview - print a view.

SYNOPSIS

prview file.v

DESCRIPTION

Object names, positions, and attributes contained in a view file are printed
This information can be useful when editing a view file.

6. Conclusions and Recommendations

A brief summary of the preliminary design contained in this report is presented. Recommendations for how to proceed in implementing the design are discussed.

6.1. Summary of the Report

This report provides a preliminary design for an intelligent human-computer interface for the TCMS at Kennedy Space Center. The report describes tools that can be used to build complex graphical objects such as menus, valuators, windows and icons from primitive parts or widgets.

One requirement for the TCMS interface is that it behave in an intelligent and consistent manner. A definition of an intelligent human-computer interface is given and methods for embedding intelligence into the objects created with the TCMS toolkit are discussed.

A concern related to the intelligence of the interface is the design of the help facility. Help provided by the TCMS interface can be quite varied as it includes not only help in using the interface, but also help for testing, monitoring and controlling the applications attached to the interface. A hypertext based help facility is proposed.

A manual describing the tools and utilities of the TCMS toolkit is provided. The tools range from simple tools which create basic objects such as lines, rectangles and text, to complex tools that create menus, windows and complete interfaces. A number of utility functions are also included in the toolkit. These utilities serve to execute an interface, collect statistics on its use, edit its attributes and perform other useful functions.

6.2. Recommendations for Implementing the Design

Implementing the TCMS interface will require time, people and other expenses. A cost estimate for the implementation needs to be prepared.

The preliminary design provided by this report should be expanded into a detailed design for the interface. Selection of commercial off-the-shelf software packages for implementing the knowledge module and the help module must be made before this detailed design can be produced.

A team of designers, programmers and administrators should be formed to implement the interface. This team should be partitioned along the lines of the three major modules of the interface: the graphics module, the knowledge module, and the help module. It is suggested that three 2 person teams be formed to implement each module. Each person in the team should be a competent programmer. The team leader should be an expert in one or more of the three identified areas. One administrator for the teams should be capable of coordinating these teams and performing other administrative duties.

Bibliography

[1] *DV-Draw Reference Manual*, V.I. Corportation, Amherst, MA, 01002, version 6.0 ed., 1988.

[2] *DV-Tools Reference Manual*, V.I. Corportation, Amherst, MA, 01002, version 6.0 ed., 1988.

[3] *DV-Tools User's Guide*, V.I. Corportation, Amherst, MA, 01002, version 6.0 ed., 1988.

[4] B. CAMPBELL AND J. M. GOODMAN, *Ham: a general purpose hypertext abstract machine*, Communications of the ACM, 31 (1988), pp. 856–861.

[5] J. CONKLIN, *Hypertext: an introduction and survey*, Computer, 20 (1987), pp. 17–41.

[6] M. RUDISILL, D. GILLIAN, ET AL., *Space Station Information System Human-Computer Interface Guide*, Tech. Rep., National Aeronautics and Space Administration, Lyndon B. Johnson Space Center, Houston, Texas, May 1988.

[7] B. SHNEIDERMANN, *Designing the User Interface*, Addison-Wesley Publishing Company, 1987.