TOOLS FOR GRAPHICAL INTERFACE DESIGN

James Hollan
Edwin Hutchins
Mark Rosenstein
Louis Weitzman

Future Technologies
Navy Personnel Research and Development Center
San Diego, CA 92152

Introduction

A major element on our research agenda is to understand the power and perspicuity of graphical interfaces. In most of our work we have been concerned with exploring the usefulness of graphics for supporting the development of understandings of complex dynamic physical systems. An important message to builders of instructional systems from cognitive science and artificial intelligence is the crucial importance of mental models, not only for understanding how experts represent knowledge but also for understanding the errors a novice might be expected to make.

We have argued (Hollan, 1984) that much of the naturalness and power of graphical interfaces results from the vast amount of experience people have in dealing with objects physically arrayed in space. Graphical interfaces can draw on this experience by providing representational elements that map more directly (than say text) to a person's mental model of a domain. In this way, the interface can make visible conceptually important aspects of a domain. In addition, it makes possible the depiction of systems with greater fidelity to the way we think and reason about them by providing multiple perspectives, maintaining explicit state information, and depicting a topology that supports and encourages the use of visualization skills.

The goal of this paper is not to reiterate our earlier arguments, but instead to survey a set of tools we have implemented to facilitate graphical interface construction. We think that these tools are examples of mechanisms that allow the combination of human and artificial intelligence and thus their discussion is germane to the topic of this conference. Before describing the tools, we would like to mention some of the ideas which motivated them. One motivation is our conviction that interactive graphical interfaces enable powerful new ways for people to interact with machines. Our experience building inspectable simulation-based training systems has convinced us of this and of the crucial importance of recognizing the variety of

types of knowledge that must be combined to build a successful system. One cannot expect to build effective instructional systems or interfaces without having a rich understanding of the domain of application. Knowledge of human cognition, the way people do and should think about the domain, must also be used to augment the domain knowledge in order to make it accessible to a student. Likewise, hardware and software technologies must be understood so that they can be effectively employed to provide productive interactions.

In order to assist in satisfying the requirements for these diverse types of knowledge, we have been building a set of tools that provide domain experts with facilities that implicitly support good interface design and allow them to do an increasing portion of the interface development task. By applying the same stringent requirements to the tools that we apply to the interface that the designer will build, we make the incorporation of the expert's knowledge into the design of the interface much more natural. If the tools allow the expert to work closer to the way she thinks about the domain, this traditionally hard task is simplified.

The types of interfaces we are interested in have sometimes been called Direct Manipulation (DM) interfaces (Shneiderman, 1982). That is an appealing term in that it captures some of the feel of the interface, but only a portion of the power of a DM interface consists of the feeling that one is directly manipulating objects on a screen. Much more important in our view is that the semantics of the task domain are built into the behavior of the interface. The entities depicted and the operations made available are those that closely match the way one wants to think about the domain. Surely the three principles Shneiderman lists are important to interface construction,

Continuous representation of the objects

Physical actions or button presses instead of complex syntax

Rapid incremental reversible operations whose impact on the object of interest is immediately visible

but the real power lies in what aspects of the domain get represented, how they are depicted, and how appropriate the operations provided are to the semantics of the domain. Thus, for example, the real power of spread-sheet programs is that actions directly associated with the semantics of the domain, like making projections, are immediately available. One changes a number and the implications of that change are automatically propagated. This allows the user to think at a level appropriate to the task, not at the level of the mechanics of arithmetic.

Our concern in designing tools to support interface construction is more with building systems for experts than with building expert systems. We would like to provide tools that allow people who are expert in a domain to build interfaces that incorporate their expertise and make it available to others. We want to construct not just tools but tools for making tools. The tools we discuss in this paper were motivated by a concern for the ways graphical interfaces can support the

development of effective mental models for reasoning and problem solving. The tools include: an object-oriented graphical editor developed to assist us in building interactive inspectable simulation-based instructional systems, a behavior editor which allows one to create simulations by putting together iconic representations of parts, and an icon editor which supports the design of new graphical icons and allows the specification of new dynamic graphical behaviors.

Graphics Editor

The Graphics Editor originated out of our work on the development of Steamer (Hollan, Hutchins, Weitzman, 1984). A major aspect of Steamer is the ability to view and interact, at a number of hierarchical levels, with a simulation of a propulsion plant via a color graphics display. For example, one can interact with the simulation at an abstract level such as is depicted in the Basic Steam Cycle diagram (Figure 1) or at a more detailed component system level, such as the Make-up and Excess Feed system (Figure 2). These dynamic graphical views consist of iconic depictions of components, gauges, and other objects designed to assist one in understanding and interacting with the plant.

It is important to understand that this graphical interface functions in two ways. First, it reflects the state of components in the simulation. Thus, for example, it reveals whether a particular component is operating or not by means of changes in color or other graphical features of the iconic representation. A pump's state is depicted by its color, appearing green if it is operating and red if it is off. Columns above valves show how far open the associated valve is. A second function of the graphical interface is to allow one to change the state of a component within the simulation model. Such changes are made by manipulating a mouse pointing device. For example, one could increase the level of a tank in Figure 2 by pointing at a high position in the tank and clicking a button on the mouse. The tank would immediately reflect the new level and more importantly the propagated effects of that change would also immediately become visible. If one were to raise the level of the tank on the left in Figure 2, it would cause the Reg Valve to open. Raising it above a certain level would cause the 1b Forward Pump to turn on. Thus, the graphical interface allows both the monitoring of the state of the plant and also its manipulation.

Diagrams are built using a Graphics Editor. An important part of a curriculum designers task, in construction of a Steamer-like interface, is to put together diagrams that assist students in understanding a simulated system. The black and white screen depicted in Figure 3 provides the designer with editing and critiquing controls, while the color screen reflects the current state of the diagram, such as seen in Figures 1 and 2, and provides control over the layout of the icons. Consider the designer building the diagram drawn in Figure 2. To lay down a tank in that diagram, the designer would select a tank icon from among the existing icon types. The cursor would immediately be moved to the color screen and the designer would choose a location for the tank by clicking a mouse button. The shape of the tank

would then be determined by moving the mouse and clicking again.

Once the diagram is completed using the tools provided by the graphics editor, the designer must relate the graphical depiction to the underlying simulation model of the 1200psi steam plant. The process of connecting icons to the simulation model is called tapping. The communication between the icons and the simulation model is through variables. An icon can probe and/or set variables. On each tick of the simulation an icon that probes a variable will reflect the value of that variable. For example, a dial would change the position of its needle to show the current value of its associated variable. An icon that sets a variable when interacted with will set the variable in the simulation model to reflect its current state. To tap the left tank in Figure 2, the designer clicks on tap on the B&W screen. A menu pops up (see Figure 4), into which the designer types the names of the appropriate variables.

Building Steamer diagrams required the existence of suitable icons to represent the physical components. It also required a math model which would simulate the actions of a steam plant. For Steamer, we have hand constructed the large set of icons. A sample of these is shown in Figure 5. We also had access to a high fidelity simulation model of a propulsion system. The goal of much of our current research is to produce tools that will relax these constraints and facilitate the construction of similar instructional system for other domains. We now turn to a discussion of two of these tools: a Behavior Editor and an Icon Editor.

Behavior Editor

One real obstacle to a domain expert being able to use the Graphics Editor to construct an interface for a domain is the requirement of an existing mathematical model of the domain. Even if a model is available it is unlikely that a domain expert, without significant additional effort, will have sufficient understanding of the simulation model to be able to tap icons into it. The Behavior Editor is an initial exploration of a tool which would eliminate the need for an underlying simulation model. The icons in the Graphics Editor know how to "appear" in order to represent the status of variables to which they are tapped, but the behavior of the system is defined by a simulation model. The Behavior Editor is composed of icons that know both the behavior and the appearance of the objects they represent. The behavior of the system emerges from the interactions of the icons with each other. The ability to incorporate the behaviors required in a simulation is facilitated by the object-oriented implementation of icons. For example, consider a dial. In the object-oriented system we use, Flavors, many of the properties of the dial actually come from component objects (Mixins, in Flavors terminology).

These components are in fact common to many of the icons. Figure 6 shows some of the components of a dial: Continuous, Rectangular and Tap mixins. Into any icon that displays continuous values, we add the object called Continuous. This object provides the icon a place to hold the icon's value and the minimum and maximum values the icon can display. The Continuous Mixin also provides

commands, or messages the dial icon will understand, such as *constrain-value* which truncates a number to be between a minimum and maximum value. Icons such as dials or valves understand other messages like *draw* which cause the receiving icon to draw itself on the color screen.

To make intelligent icons capable of generating a simulation, it is necessary to add components with domain and simulation knowledge. We have built a number of icons that know the rudiments of fluid dynamics and understand about connections to other icons. These include tanks and pipes that know about pressure and flow, so they can be used in fluid systems. Figure 7 depicts a very simple system constructed in exactly the same manner as with the Graphics Editor but involving behaviorally "smart" icons that can be connected together. These icons have mechanisms for recognizing when connections should be made and thus the topology of the diagram is automatically generated.

An excellent example of the flexibility engendered by icons with connectivity knowledge is the Super Sensor. This icon is a dial which asks icons it connects to what variables can be monitored and which variable to measure by default. In Figure 7 notice that a Super Sensor is connected to the middle tank, to monitor its level. Sensors know how to monitor appropriate aspects of the objects to which they are connected. We can manipulate these aspects by clicking on the *Miscellaneous* menu item on the Behavior Editor screen. In Figure 8 we see that the sensor has defaulted to measure *value*, which is highlighted. If the designer wishes, he can change the sensor to measure fluid exchanged by clicking on that menu item. When we interact with the tanks, setting their levels, we are actually setting the initial conditions of the simulation. On the Behavior Editor screen (Figure 8), we can then click on run, and the simulation will run till equilibrium. It is important to notice a fundamental difference between the Behavior Editor and the Graphics Editor. In the Graphics Editor, the designer would need a mathematical model of these tanks and pipes and then it would be necessary to find the appropriate variables to tap the icons into. Here the icons themselves perform the required computations for the simulation.

All these icons, both those with and without domain knowledge, were hand coded. It became clear to us that while these icons are useful in this particular domain, we required a tool to help expand the range of domains by facilitating the building of new icons. We have begun the construction of an editor to assist in creating new icons without requiring the user to resort to traditional programming.

Icon Editor

The development of an Icon Editor is one of our current projects. The interface to this editor is depicted in Figure 9. From a set of basic icons and behaviors a designer with knowledge of the fundamentals of a domain can construct icons, both for Behavior Editor applications and also for use with the Graphics Editor when a simulation is available. The Icon Editor provides an environment in which complex icons can be constructed from more basic ones. In addition, we also

provide mechanisms for a user to specify new behaviors for icons. The pane labeled *Existing Icons* in Figure 9 shows the set of icons an icon builder can choose from. As component pieces of the icon are added, their presence is reflected in the *Constituents* list. Procedures for the new icon are constructed in the ***Method*** area.

One of our current interests is in understanding the task of creating an intelligent appointment planner. As part of that work, it was necessary to construct calendars. A calendar's basic unit is a date object. Its use in an intelligent calendar would be to interact with other calendars to resolve scheduling conflicts. Thus, a person might instruct the calendar that she would like to meet with another individual. The calendars of the two individuals would then negotiate an agreeable date. The date objects would also have a graphical form to provide a means of visually displaying the calendar. Here we consider how that image would be built up and how a simple behavior for the icon would be constructed.

The image of the date is a box with the day and the numerical date. An icon builder, first specifies the components of the date icon. From the list of existing icons, a rectangle is selected as the outline for the date icon. Clicking on name, a menu pops up with the default name Rectangle. "Background" is then typed into the name slot, allowing this icon to be referred to with a meaningful name. In a similar way, we can lay out a day and numerical date in the date icon. These are also given names and default values and colors. These are the constituents listed in the *Constituents Pane* of Figure 9.

We are currently exploring methods that allow the icon builder to specify behavior. To continue with our example consider an icon builder specifying the behavior to change the day. The builder selects *Method* in the menu and types in the name of this message: *set-day*. A menu pops up allowing the specification of day as the argument to the message. The three steps in changing the day are to have the day erase itself, set its string to be the new day, and then draw itself. Clicking on *Transform* allows the specification of the first step in the procedure. The builder clicks on day, and then types in the message *erase*, which is what the day should do first. Clicking on *Do It* installs this step. Similarly the other two steps are defined. These are the steps shown in the *Transforms Pane* of Figure 9. To test the procedure, we might click on send. A menu would pop up asking us to specify what day. We might type in *Thursday*, and click on *Send*. On the color screen, the day would change to Thursday, providing immediate feedback that this procedure is correct. In this way behaviors for the icon can be specified. Our first experiments allow specification through defaults and pop up menus. In later versions, we expect more and more of this specification will be performed graphically.

Future Work

Currently we are building views into real-time systems, such as the view shown in Figure 10 which monitors activity on a VAX 780 running Unix. A statistical assistant and a scheduling assistant are also under construction. Other support tools are in the planning stages. One of these is a Graphics Design Assistant to help a domain expert better utilize knowledge from the graphic arts to provide views that are visually more consistent and informative. An instructional support tool is also being planned which will allow the domain expert to specify activities and tests based around a diagram or a series of diagrams.

Conclusion

In this paper, we have briefly discussed three tools: the Graphics Editor, the Behavior Editor, and the Icon Editor. These tools greatly facilitate the construction of interactive graphical interfaces. The ability of students and researchers to see into systems increases their ability to understand and control those environments.

A particularly appealing aspect of the theme of this conference, *Combining Human and Artificial Intelligence: A New Frontier for Human Factors*, is that it may lead us to look beyond the current fascination with expert systems to much more exciting forms of interaction, interactions based on the understanding that the computer is more than a new tool: it is a new medium for the construction of tools.

References

Hollan, J. D. (1984). Intelligent Object-Based Graphical Interfaces. *Human-Computer Interaction*, G. Salvendy (Ed.), Elsevier, Amsterdam, 293-297.

Hollan, J. D., Hutchins, E. L., & Weitzman, L. (1984). STEAMER: An Interactive Inspectable Simulation-Based Training System. *AI Magazine*, 5 (2), 15-27.

Shneiderman, B. (1982). The Future of Interactive Systems and Emergence of Direct Manipulation, *Behavior and Information Technology*, 1, 237-256.

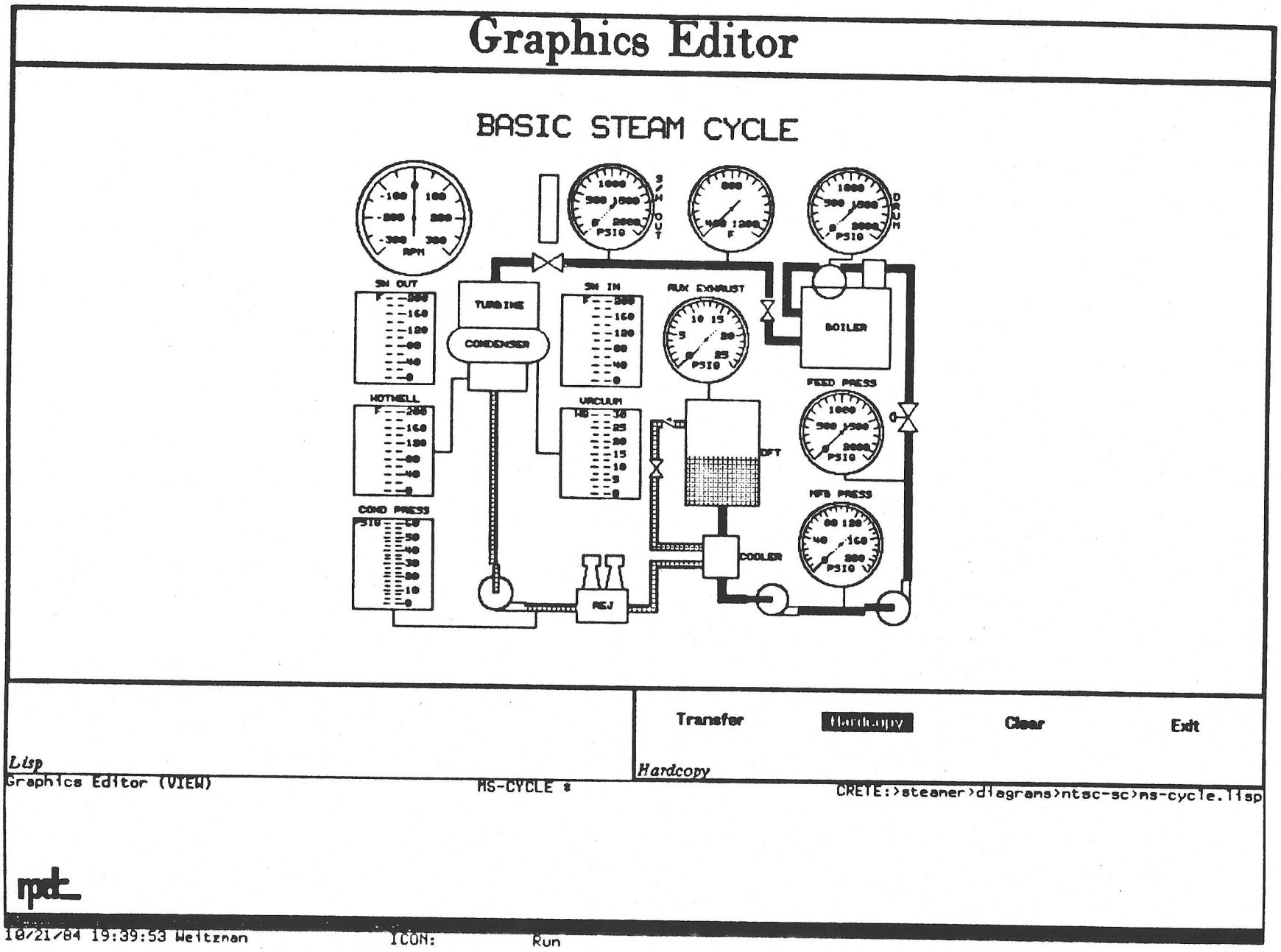
Graphics Editor
BASIC STEAM CYCLE
TURBINE
CONDENSER
BOILER
COOLER
REJ
DFT
AUX EXHAUST
VACUUM
FEED PRESS
MFB PRESS
COND PRESS
HOTWELL
Transfer
Hardcopy
Clear
Exit
Lisp
Graphics Editor (VIEW)
MS-CYCLE *
CREATE:>steaner>diagrams>ntsc-sc>ms-cycle.lisp
10/21/84 19:39:53 Weitzman
ICON:
Run

FIGURE 1

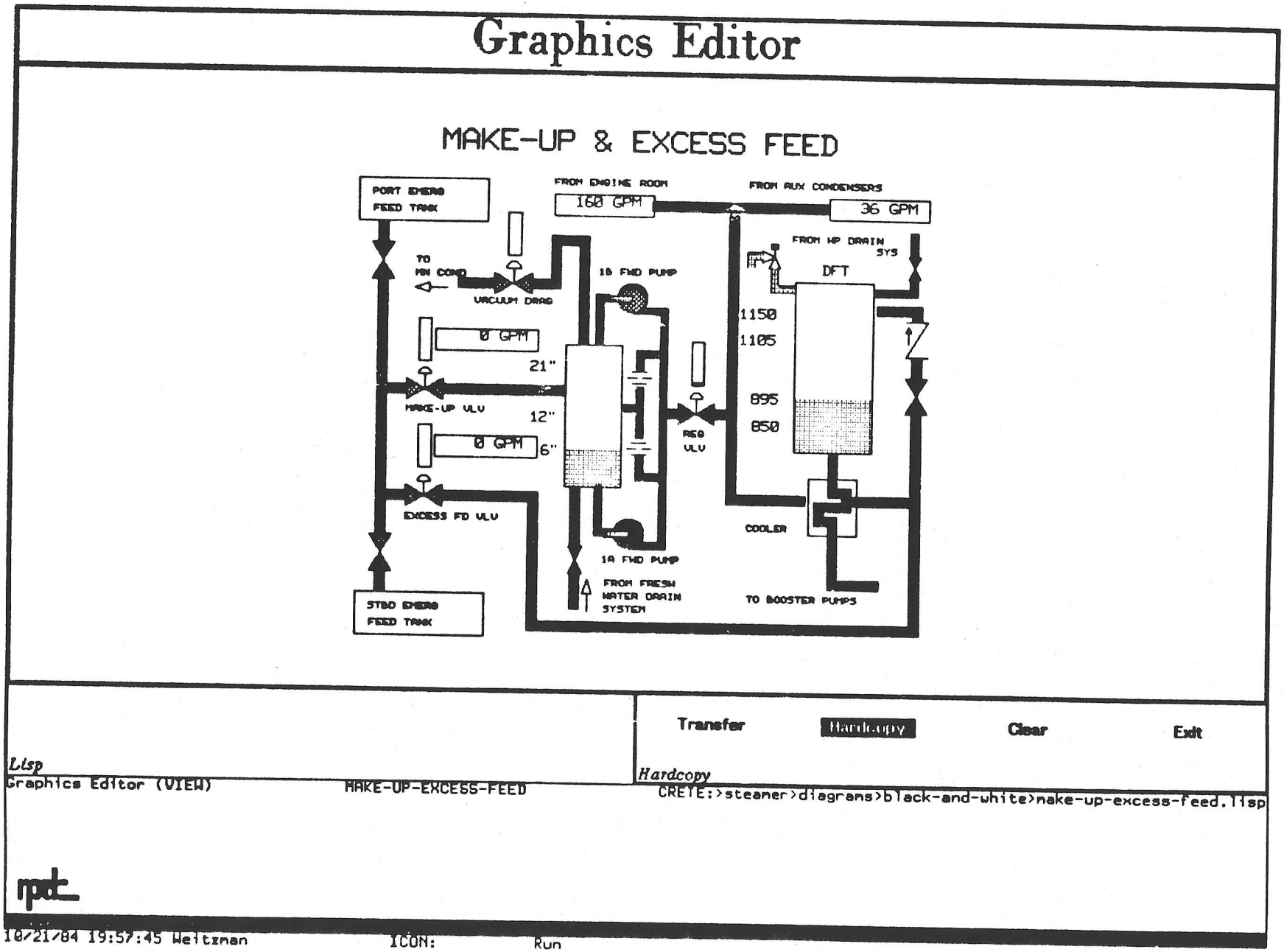
Graphics Editor
MAKE-UP & EXCESS FEED
PORT EMERG FEED TANK
FROM ENGINE ROOM
160 GPM
FROM AUX CONDENSERS
36 GPM
FROM MP DRAIN SYS
DFT
TO MN COND
VACUUM DRAG
1B FWD PUMP
0 GPM
21"
MAKE-UP VLV
12"
0 GPM
6"
REG VLV
1150
1105
895
850
EXCESS FD VLV
COOLER
1A FWD PUMP
FROM FRESH WATER DRAIN SYSTEM
TO BOOSTER PUMPS
STBD EMERG FEED TANK
Transfer
Hardcopy
Clear
Exit
Lisp
Graphics Editor (VIEW)
MAKE-UP-EXCESS-FEED
Hardcopy
CREIE:>steamer>diagrams>black-and-white>make-up-excess-feed.lisp
rpdc
10/21/84 19:57:45 Weitzman
ICON:
Run

FIGURE 2

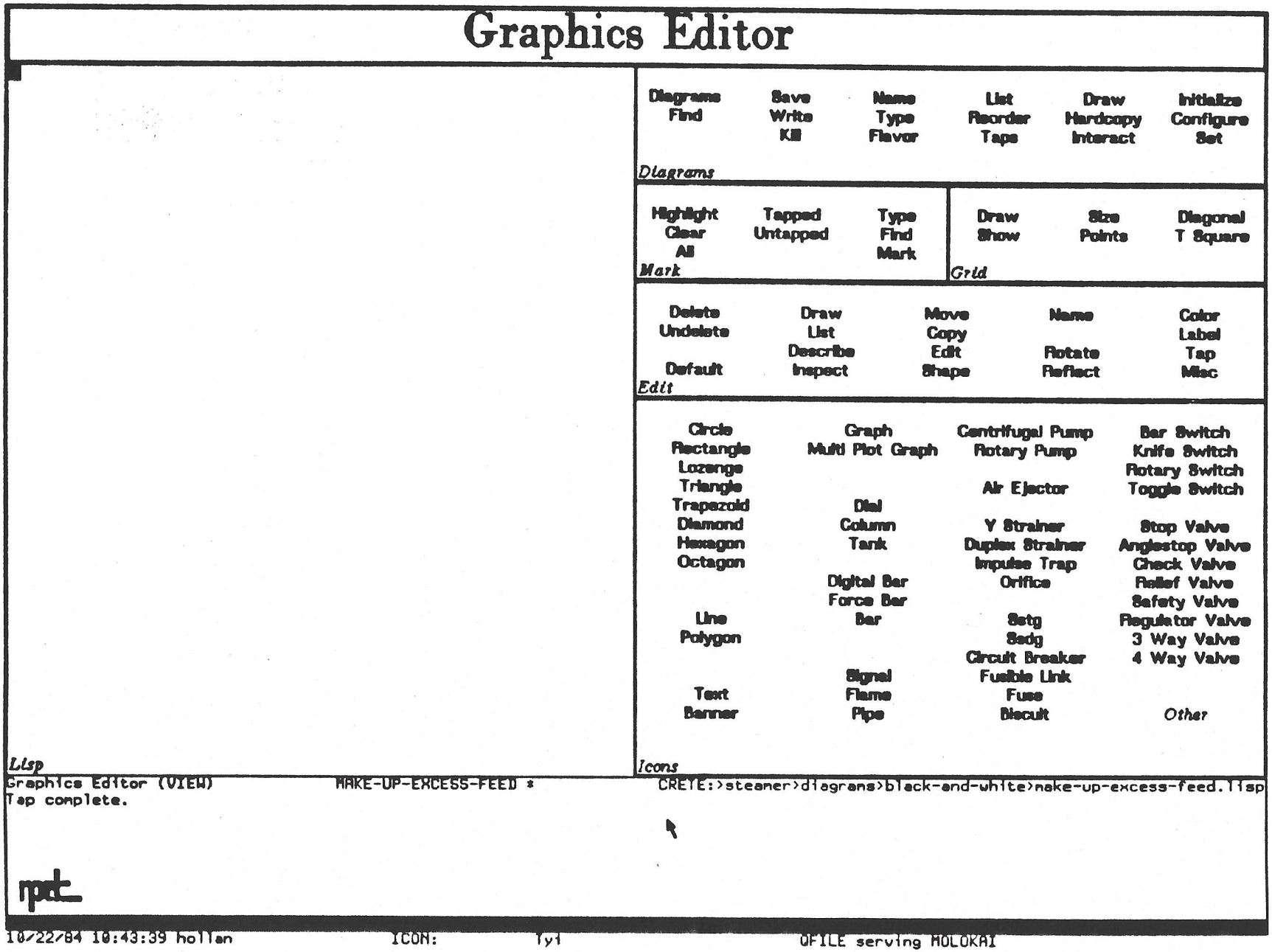
Graphics Editor
Diagrams
Find
Save
Write
Kill
Name
Type
Flavor
List
Recorder
Tape
Draw
Hardcopy
Interact
Initialize
Configure
Set
Mark
Highlight
Clear
All
Tapped
Untapped
Type
Find
Mark
Grid
Draw
Show
Size
Points
Diagonal
T Square
Edit
Delete
Undelete
Default
Draw
List
Describe
Inspect
Move
Copy
Edit
Shape
Name
Rotate
Reflect
Color
Label
Tap
Misc
Icons
Circle
Rectangle
Lozenge
Triangle
Trapezoid
Diamond
Hexagon
Octagon
Line
Polygon
Text
Banner
Graph
Multi Plot Graph
Dial
Column
Tank
Digital Bar
Force Bar
Bar
Signal
Flame
Pipe
Centrifugal Pump
Rotary Pump
Air Ejector
Y Strainer
Duplex Strainer
Impulse Trap
Orifice
Satg
Sadg
Circuit Breaker
Fusible Link
Fuse
Biscuit
Bar Switch
Knife Switch
Rotary Switch
Toggle Switch
Stop Valve
Anglestop Valve
Check Valve
Relief Valve
Safety Valve
Regulator Valve
3 Way Valve
4 Way Valve
Other
Lisp
Graphics Editor (VIEW)
MAKE-UP-EXCESS-FEED *
CREIE:>steaner>diagrans>black-and-white>nake-up-excess-feed.lisp
Tap complete.
rpdc
10/22/84 10:43:39 hollan
ICON:
Tyi
OFILE serving MOLOKAI

FIGURE 3

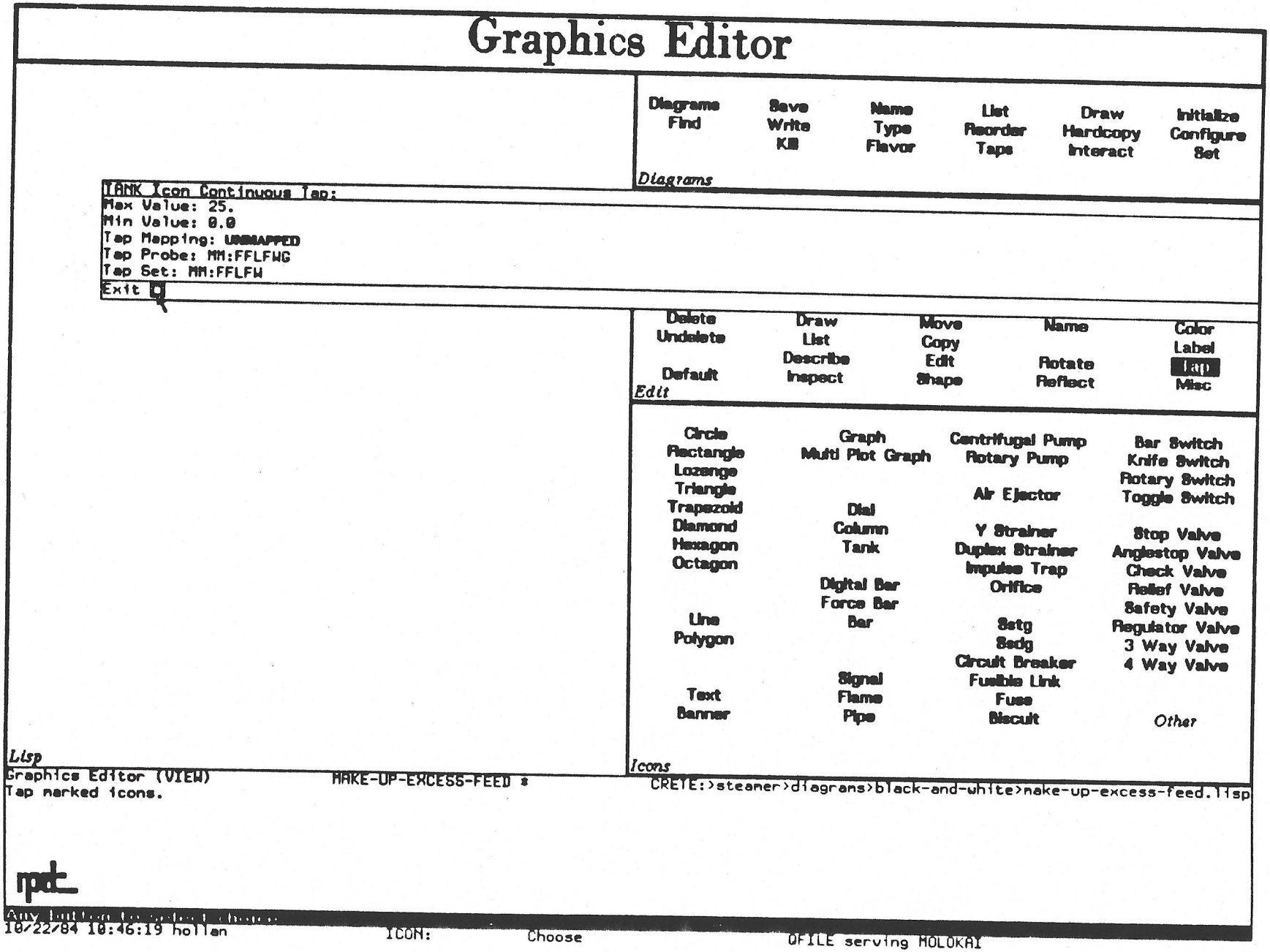
Graphics Editor
Diagrams
Find
Save
Write
Kill
Name
Type
Flavor
List
Reorder
Taps
Draw
Hardcopy
Interact
Initialize
Configure
Set
TANK Icon Continuous Tap:
Max Value: 25.
Min Value: 0.0
Tap Mapping: UNMAPPED
Tap Probe: MM:FFLFWG
Tap Set: MM:FFLFW
Exit
Edit
Delete
Undelete
Default
Draw
List
Describe
Inspect
Move
Copy
Edit
Shape
Name
Rotate
Reflect
Color
Label
Tap
Misc
Icons
Circle
Rectangle
Lozenge
Triangle
Trapezoid
Diamond
Hexagon
Octagon
Line
Polygon
Text
Banner
Graph
Multi Plot Graph
Dial
Column
Tank
Digital Bar
Force Bar
Bar
Signal
Flame
Pipe
Centrifugal Pump
Rotary Pump
Air Ejector
Y Strainer
Duplex Strainer
Impulse Trap
Orifice
Satg
Sadg
Circuit Breaker
Fusible Link
Fuse
Biscuit
Bar Switch
Knife Switch
Rotary Switch
Toggle Switch
Stop Valve
Anglestop Valve
Check Valve
Relief Valve
Safety Valve
Regulator Valve
3 Way Valve
4 Way Valve
Other
Lisp
Graphics Editor (VIEW)
MAKE-UP-EXCESS-FEED s
CREYE:>steaner>diagrans>black-and-white>nake-up-excess-feed.lisp
Tap marked icons.
10/22/84 10:46:19 hollan
ICON:
Choose
QFILE serving MOLOKAI

FIGURE 4

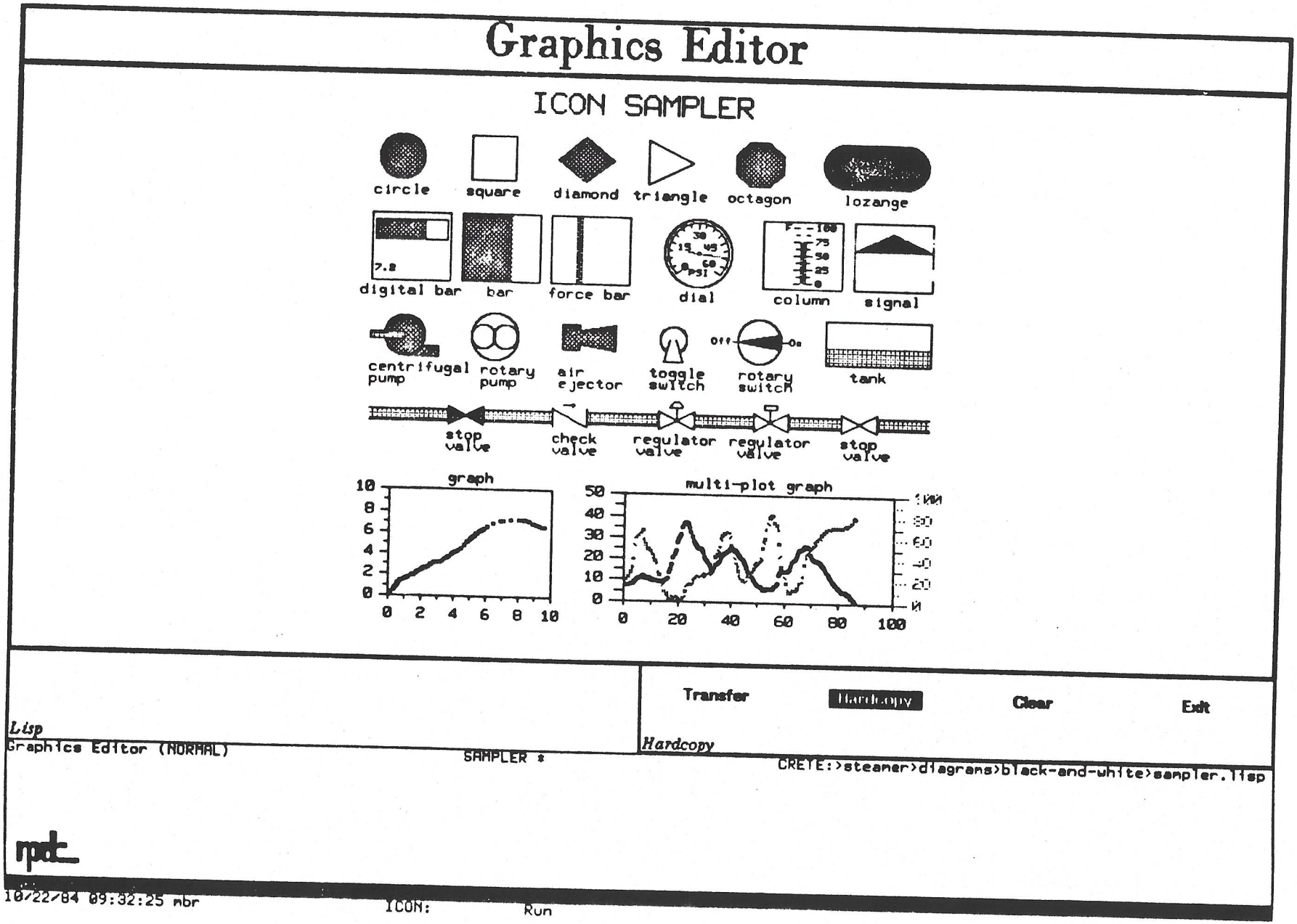
Graphics Editor
ICON SAMPLER
circle
square
diamond
triangle
octagon
lozange
digital bar
bar
force bar
dial
column
signal
centrifugal pump
rotary pump
air ejector
toggle switch
rotary switch
tank
stop valve
check valve
regulator valve
regulator valve
stop valve
graph
multi-plot graph
Transfer
Hardcopy
Clear
Exit
Lisp
Graphics Editor (NORMAL)
SAMPLER *
Hardcopy
CREIE:>steaner>diagrans>black-and-white>sanpler.lisp
10/22/84 09:32:25 nbr
ICON:
Run

FIGURE 5

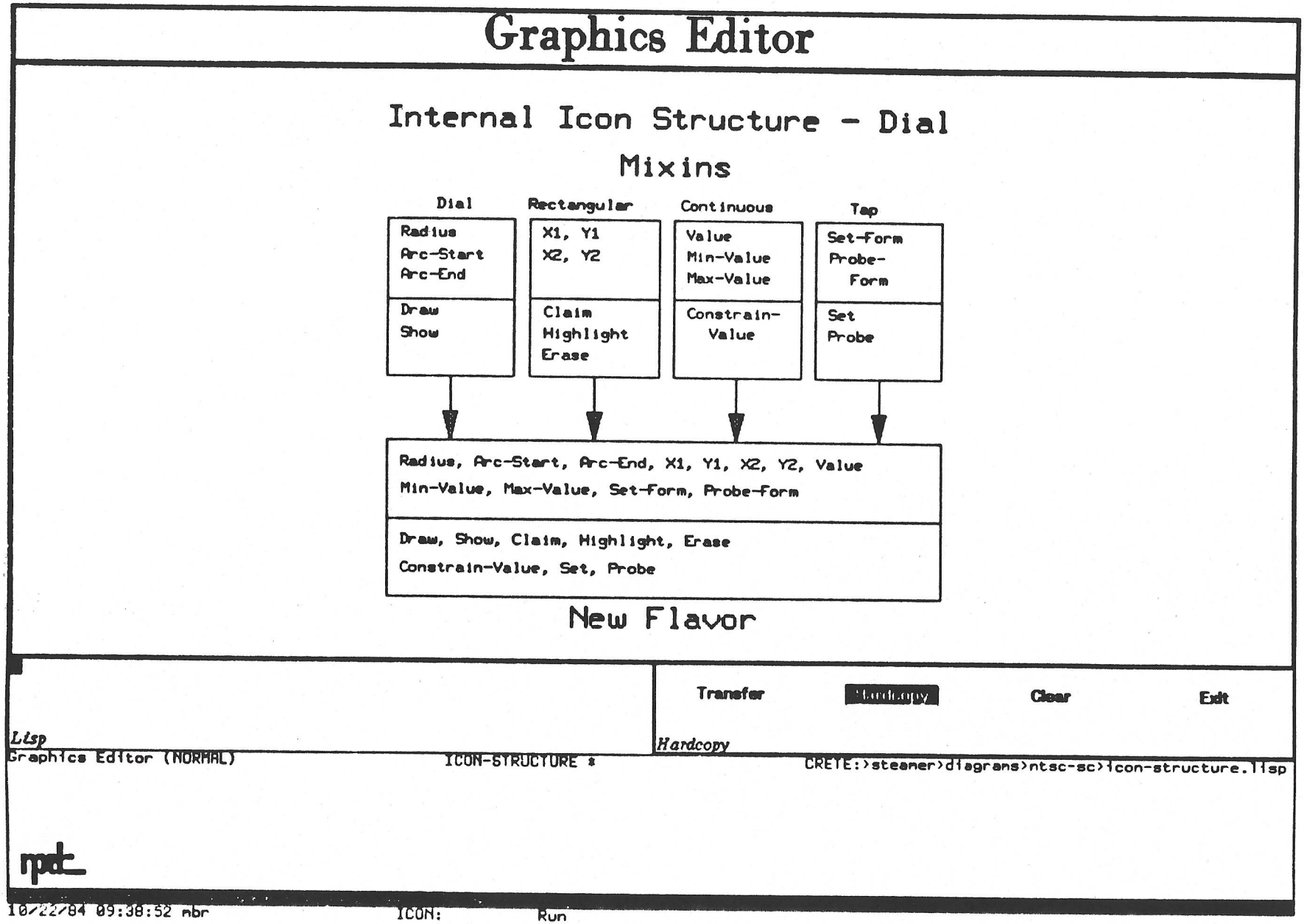
Graphics Editor
Internal Icon Structure - Dial
Mixins
Dial
Radius
Arc-Start
Arc-End
Draw
Show
Rectangular
X1, Y1
X2, Y2
Claim
Highlight
Erase
Continuous
Value
Min-Value
Max-Value
Constrain-
Value
Tap
Set-Form
Probe-
Form
Set
Probe
Radius, Arc-Start, Arc-End, X1, Y1, X2, Y2, Value
Min-Value, Max-Value, Set-Form, Probe-Form
Draw, Show, Claim, Highlight, Erase
Constrain-Value, Set, Probe
New Flavor
Transfer
Hardcopy
Clear
Exit
Lisp
Hardcopy
Graphics Editor (NORMAL)
ICON-STRUCTURE *
CREYE:>steaner>diagrans>ntsc-sc>icon-structure.lisp
10/22/84 09:38:52 nbr
ICON:
Run

FIGURE 6

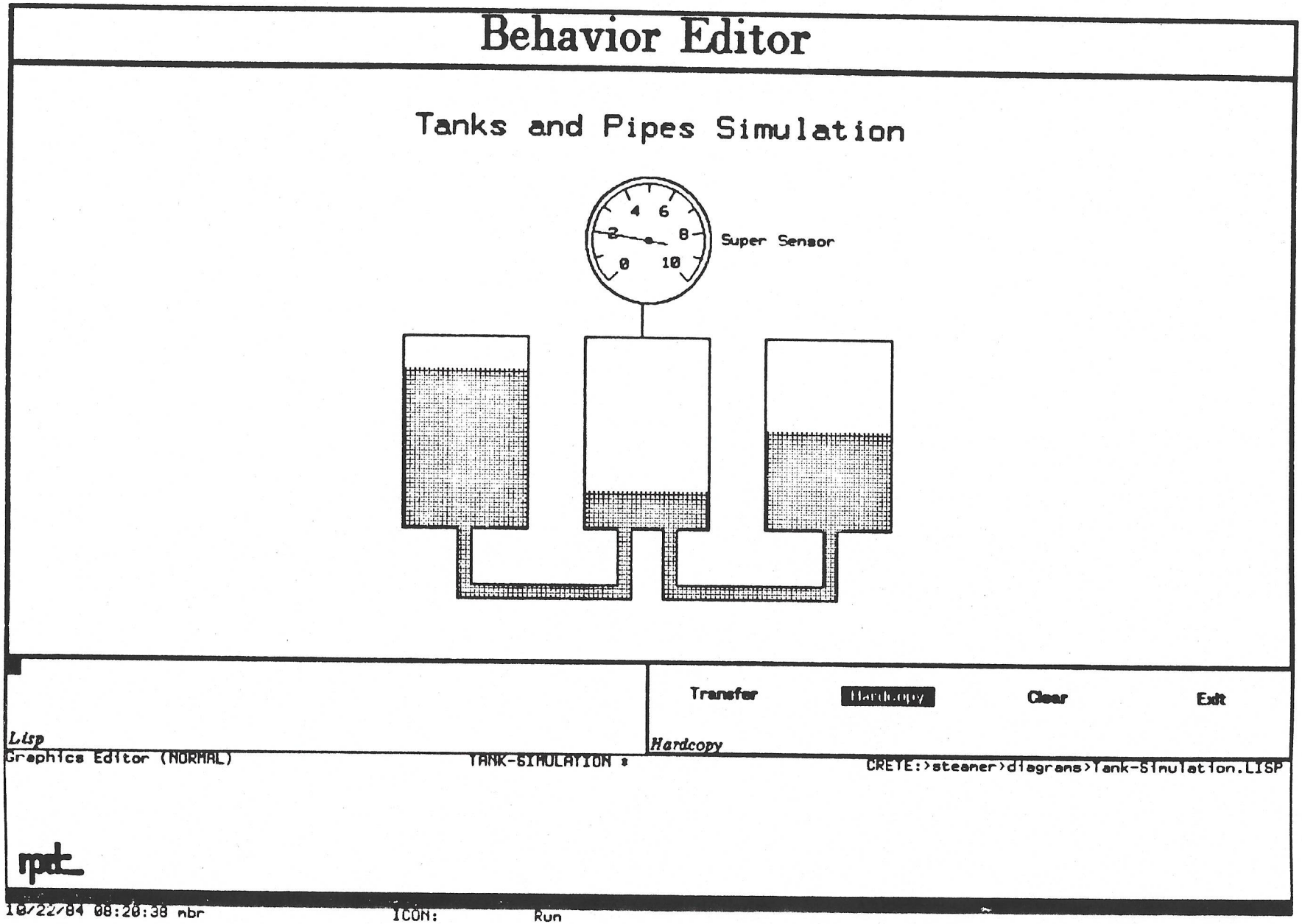
Behavior Editor
Tanks and Pipes Simulation
4
6
2
8
0
10
Super Sensor
Transfer
Hardcopy
Clear
Exit
Lisp
Hardcopy
Graphics Editor (NORMAL)
TANK-SIMULATION
CREATE:>steaner>diagrans>tank-simulation.LISP
10/22/84 08:20:38 nbr
ICON:
Run

FIGURE 7

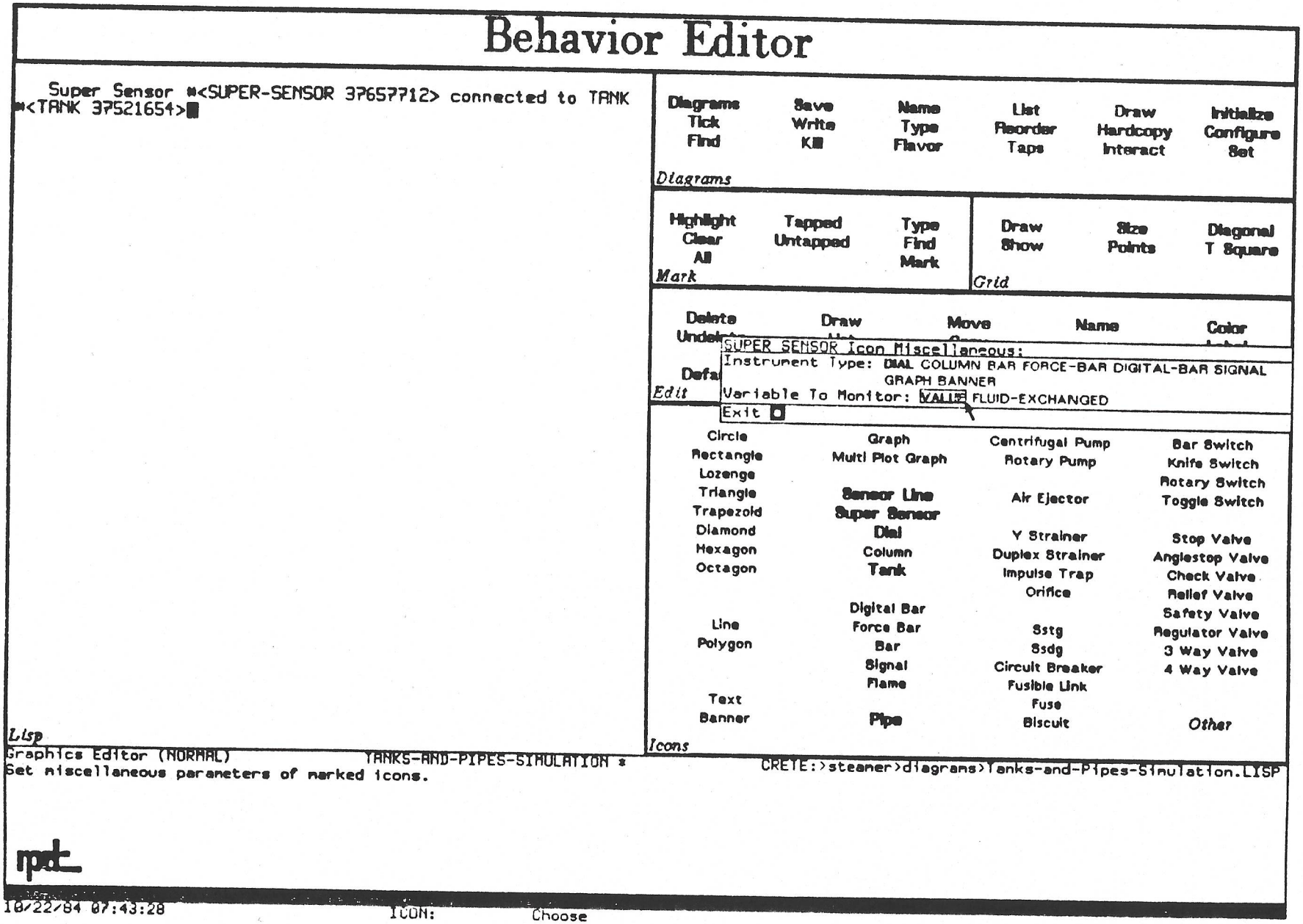
Behavior Editor
Super Sensor #<SUPER-SENSOR 37657712> connected to TANK #<TANK 37521654>
Diagrams Tick Find
Save Write Kill
Name Type Flavor
List Reorder Taps
Draw Hardcopy Interact
Initialize Configure Set
Diagrams
Highlight Clear All
Tapped Untapped
Type Find Mark
Mark
Draw Show
Size Points
Diagonal T Square
Grid
Delete Undel
Draw
Move
Name
Color
Defa
Edit
SUPER SENSOR Icon Miscellaneous:
Instrument Type: DIAL COLUMN BAR FORCE-BAR DIGITAL-BAR SIGNAL GRAPH BANNER
Variable To Monitor: VALVE FLUID-EXCHANGED
Exit
Circle
Rectangle
Lozenge
Triangle
Trapezoid
Diamond
Hexagon
Octagon
Line
Polygon
Text
Banner
Graph
Multi Plot Graph
Sensor Line
Super Sensor
Dial
Column
Tank
Digital Bar
Force Bar
Bar
Signal
Flame
Pipe
Centrifugal Pump
Rotary Pump
Air Ejector
Y Strainer
Duplex Strainer
Impulse Trap
Orifice
Sstg
Ssdg
Circuit Breaker
Fusible Link
Fuse
Biscuit
Bar Switch
Knife Switch
Rotary Switch
Toggle Switch
Stop Valve
Anglestop Valve
Check Valve
Relief Valve
Safety Valve
Regulator Valve
3 Way Valve
4 Way Valve
Other
Icons
Lisp
Graphics Editor (NORMAL) TANKS-AND-PIPES-SIMULATION *
Set miscellaneous parameters of marked icons.
CRETE:>steaner>diagrams>Tanks-and-Pipes-Simulation.LISP
10/22/84 07:43:28
ICON:
Choose

FIGURE 8

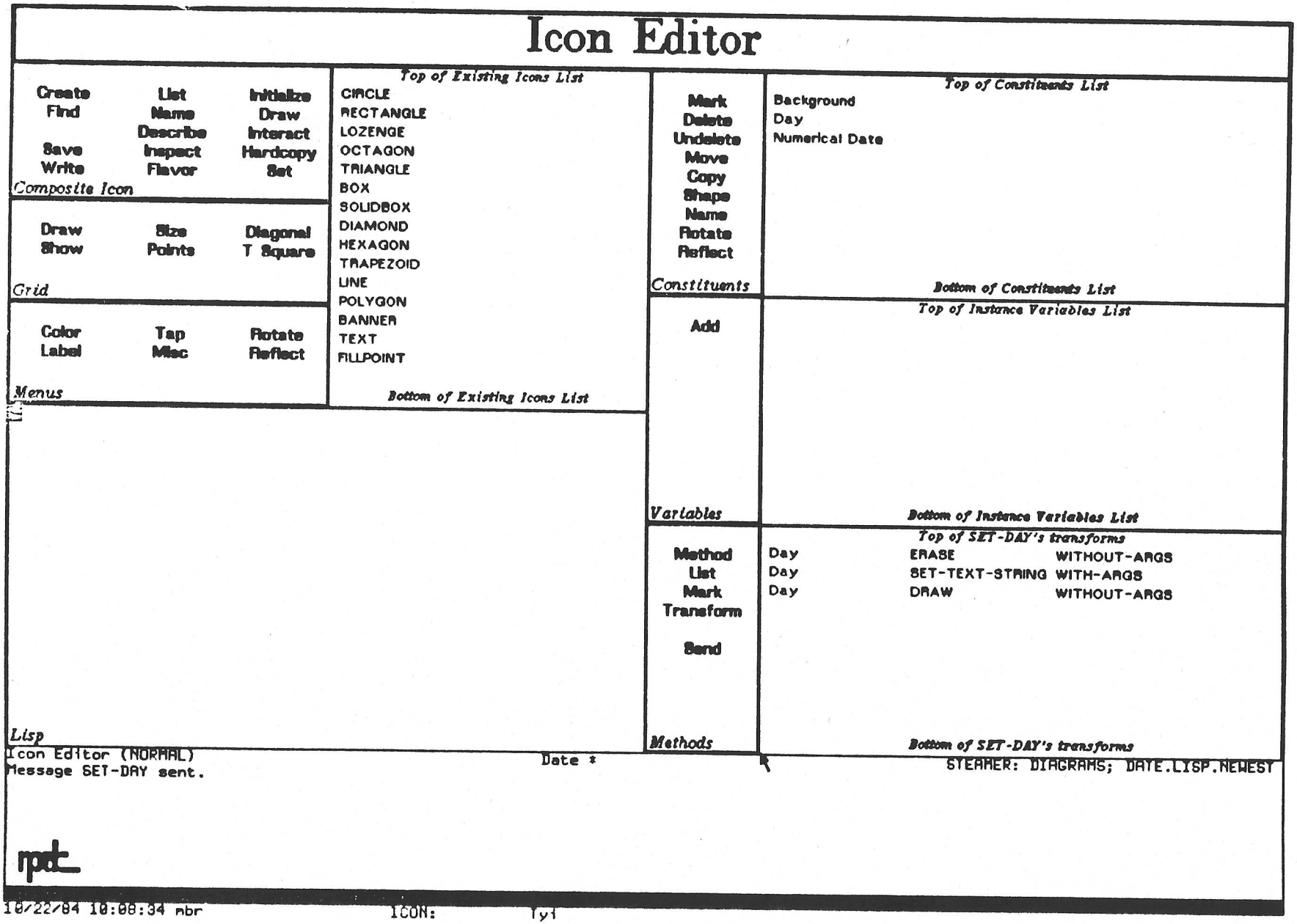
Icon Editor
Create
Find
Save
Write
List
Name
Describe
Inspect
Flavor
Initialize
Draw
Interact
Hardcopy
Set
Composite Icon
Draw
Show
Size
Points
Diagonal
T Square
Grid
Color
Label
Tap
Misc
Rotate
Reflect
Menus
Top of Existing Icons List
CIRCLE
RECTANGLE
LOZENGE
OCTAGON
TRIANGLE
BOX
SOLIDBOX
DIAMOND
HEXAGON
TRAPEZOID
LINE
POLYGON
BANNER
TEXT
FILLPOINT
Bottom of Existing Icons List
Lisp
Mark
Delete
Undelete
Move
Copy
Shape
Name
Rotate
Reflect
Constituents
Top of Constituents List
Background
Day
Numerical Date
Bottom of Constituents List
Add
Variables
Top of Instance Variables List
Bottom of Instance Variables List
Method
List
Mark
Transform
Send
Methods
Top of SET-DAY's transforms
Day ERASE WITHOUT-ARGS
Day SET-TEXT-STRING WITH-ARGS
Day DRAW WITHOUT-ARGS
Bottom of SET-DAY's transforms
Icon Editor (NORMAL)
Message SET-DAY sent.
Date *
STEAMER: DIAGRAMS; DATE.LISP.NEWEST
10/22/84 10:08:34 nbr
ICON:
Tyi

FIGURE 9

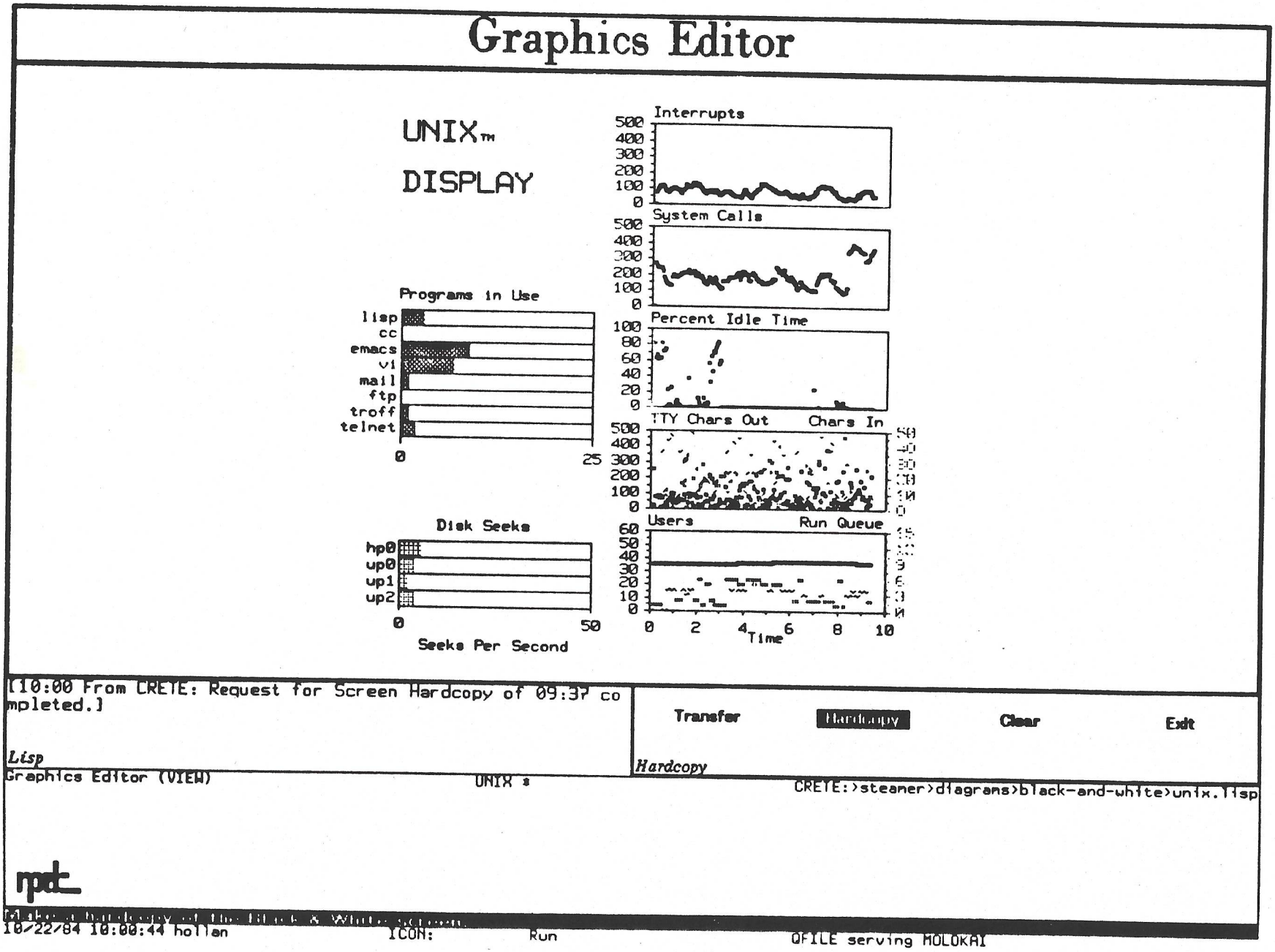
Graphics Editor
UNIX™
DISPLAY
Interrupts
System Calls
Percent Idle Time
TTY Chars Out
Chars In
Users
Run Queue
Time
Programs in Use
lisp
cc
emacs
vi
mail
ftp
troff
telnet
Disk Seeks
hp0
up0
up1
up2
Seeks Per Second
[10:00 From CRETE: Request for Screen Hardcopy of 09:37 completed.]
Transfer
Hardcopy
Clear
Exit
Lisp
Hardcopy
Graphics Editor (VIEW)
UNIX
CRETE:>steaner>diagrans>black-and-white>unix.lisp
10/22/84 10:00:44 hollan
ICON:
Run
QFILE serving MOLOKAI

FIGURE 10