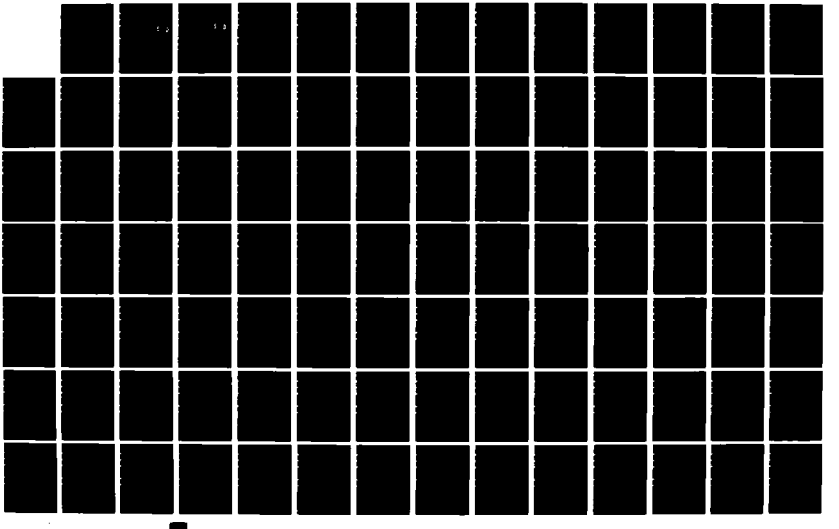
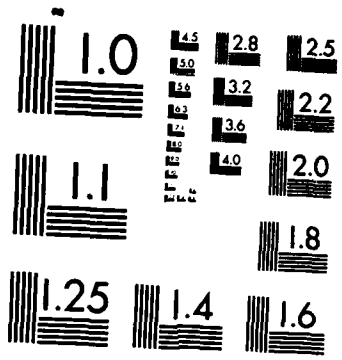
AD-A164 122 A PROGRAMMER'S ASSISTANT FOR A SPECIAL-PURPOSE DATAFLOW LANGUAGE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING A J BLACK DEC 85 1/2

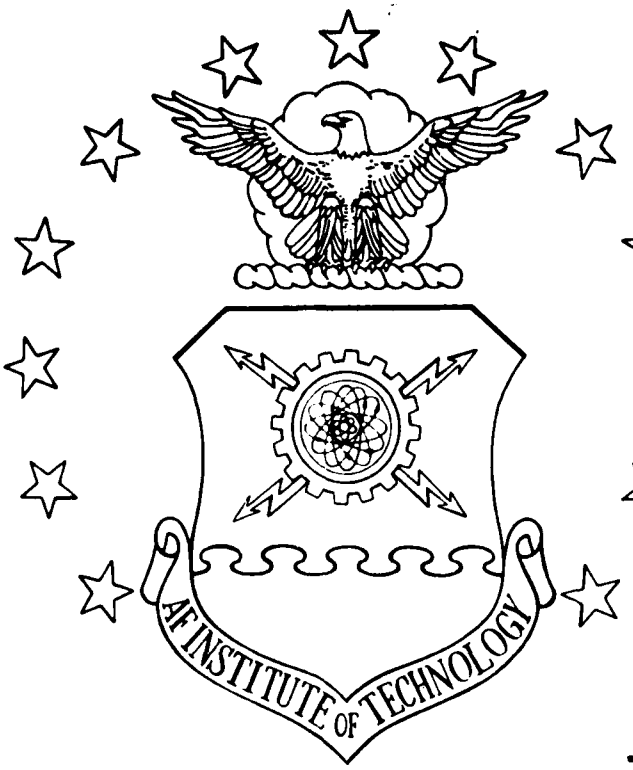
UNCLASSIFIED AFIT/GCS/ENG/85D-2 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A164 122



DTIC
ELECTE
FEB 13 1986
S D
D

A PROGRAMMER'S ASSISTANT FOR A SPECIAL-PURPOSE DATAFLOW LANGUAGE

THESIS

Alan J. Black
Captain, USAF

AFIT/GCS/ENG/85D-2

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

DTIC FILE COPY

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

86 2 12 075

AFIT/GCS/ENG/85D

DTIC
ELECTE
FEB 1 3 1986
S D
D

A PROGRAMMER'S ASSISTANT FOR A SPECIAL-PURPOSE DATAFLOW LANGUAGE

THESIS

Alan J. Black
Captain, USAF

AFIT/GCS/ENG/85D-2

Approved for public release; distribution unlimited

AFIT/GCS/ENG/85D

A PROGRAMMER'S ASSISTANT FOR A SPECIAL-PURPOSE DATAFLOW LANGUAGE

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electical Engineering

Alan J. Black, B.S.
Captain, USAF

December 1985

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

Approved for public release; distribution unlimited

QUALITY INSPECTED 8

Acknowledgments

I want to thank my thesis advisor, Dr. Gary B. Lamont, for his help and advice. A special thanks is extendend to Captain Steven K. Rogers for his help in defining the requirements for this project. Finally, I want to express my appreciation to my thesis committee, Captain Steven Cross, Dr. Matthew Kabrisky, and Professor Charles W. Richard.

Table of Contents

List of Figures

AFIT/GCS/ENG/85D

Abstract

A programming tool, the Graph Design Assistant (GDA), for a special-purpose dataflow language was designed and implemented. The motivation for the effort was the need to construct a robot simulation facility which will assist in the development of effective algorithms to plan and control robot movements.

An Evans and Sutherland PS300 graphic workstation is used to display animated robot simulations. The Graph Design Assistant was developed so that researchers could program robot simulations on the PS300 without having to learn the intricacies of the PS300's dataflow language.

The Graph Design Assistant was implemented on a "Lisp machine" using a knowledge engineering tool. The result of the effort was a prototype system to be used as the basis for further development.

A PROGRAMMER'S ASSISTANT FOR A SPECIAL-PURPOSE DATAFLOW LANGUAGE

I. Introduction

This thesis describes an investigation into the methodology of computer programming. There is a critical need to increase the power of our programming tools. John Backus has stated "there is a desperate need for a powerful methodology to help us think about programs, and no conventional language even begins to meet that need" (Backus, 1978:614).

This thesis deals with a specialized problem, programming a graphic work station using a special-purpose dataflow language. The result of the thesis effort was the development of the Graph Design Assistant (GDA). The system acts as a programmer's assistant, which guides and advises the programmer as he interactively builds a dataflow program.

Although GDA was intended to solve a rather specific problem, the experience gained and the resulting system is applicable to many complex design tasks.

1.1 Background--A Robot Simulation Facility

The motivation for this thesis effort is to provide a robot simulation facility for the Air Force Institutes of Technology's Information Sciences Laboratory. The ability to simulate a robot is important for the development of robotic planning and control algorithms, the development of new robots, and for off-line programming of existing robots.

Current robots can perform well defined tasks, but perform poorly in uncontrolled environments (Brady 1985:80-81). A robot simulation can be useful in the development of robotic control and planning algorithms to deal with complex situations.. A simulation facility would make it possible to test algorithms without having to actually build a physical robot.

A robot simulation facility can be used in the development of new robots. Different designs can be tested before the actual robot is constructed.

A simulation facility is even useful for existing robots. Control programs for existing robots can be developed and tested off-line (independent of the existing robot). This capability will be important in industry because it can prevent losses caused by removing an expensive robot from service in order to program it (Thomsom, 1984:335-336). In addition, because software errors may cause hazardous situations with a real robot, a

simulation can be used to thoroughly test new software in a safe environment (Pinson, 1985).

1.2 Problem

The specific problem addressed by this thesis effort is the development of a programming tool for constructing special-purpose dataflow programs. The dataflow language is a special-purpose language used to program an Evans and Sutherland PS300 graphic workstation.

Thus, the objective is to allow researchers to program a robot simulation on the PS300 without having to learn the intricacies of the PS300's dataflow language.

1.3 Scope

The task of producing a general purpose automatic programming system is difficult. Most of the existing efforts are experimental and are capable of solving small problems (Partsch, 1983: 229). One effort has been underway for 15 years and has just recently produced an "operational testbed" (Balzer, 1985:1257). Rather than attempting to fulfill "that ever receding goal of automating the programming of everything the user wants with a minimal amount of specification" (Kant, 1985:1371), the scope of the Graph Design Assistant was limited in several ways.

The product of GDA is a program in the special-purpose dataflow language. Dataflow programs are easily represented with graphs; therefore, a graphic representation was devised for the user to manipulate.

The application domain has been limited to producing robot simulations. This restriction is not unreasonable, in fact, it has been suggested that in order for an automatic programming system to benefit the "computationally naive user", the system must contain domain-specific knowledge (Barstow, 1985:1321).

Another limitation on the scope of the effort is that the initial version of the Graph Design Assistant is viewed as a prototype. One view of prototyping is that its purpose is to produce a specification for a system (Floyd, 1984:9). As such, this effort was to produce a complete "outline" of the system that will demonstrate feasibility and serve as a specification for further development.

1.5 General Approach

The approach in the development of the Graph Design Assistant was to focus on the construction of an interactive tool that is a "mix of human and machine power" (Kant, 1985:1373). The system is was designed to fit somewhere between the extremes of conventional compilers and fully automatic programming systems.

In the interactive environment of the Graph Design Assistant, the user specifies a program by building a graph on the screen. As the graph is built, GDA checks the "consistency" of the graph and notifies the user of any problems.

With GDA, much of the programming process involves the selection of "plans" from a library. The term plan is used to mean a "standard form" (Rich, 1981:1044). A plan is instantiated and then expanded (like a macro) to form a portion of the graph. At the top of Figure 1.1 is a graph network with nodes N1, N2, N3, and N5. The user has selected a plan PLAN-4 a placed it in the network. When the plan is instantiated, it "unfolds" to create the graph at the bottom of Figure 1.1.

There are many different approaches to the development of programming tools. Some approaches are similar but are described with different terms by their designers. What follows is a comparison and contrast between several approaches and the Graph Design Assistant.

Conventional programming tools. Conventional programming tools consist of high level compilers and interpreters, and utilities such as program editors, library managers, debuggers, etc. Of all the approaches that will be discussed, the conventional approach relies most on the user's decisions.

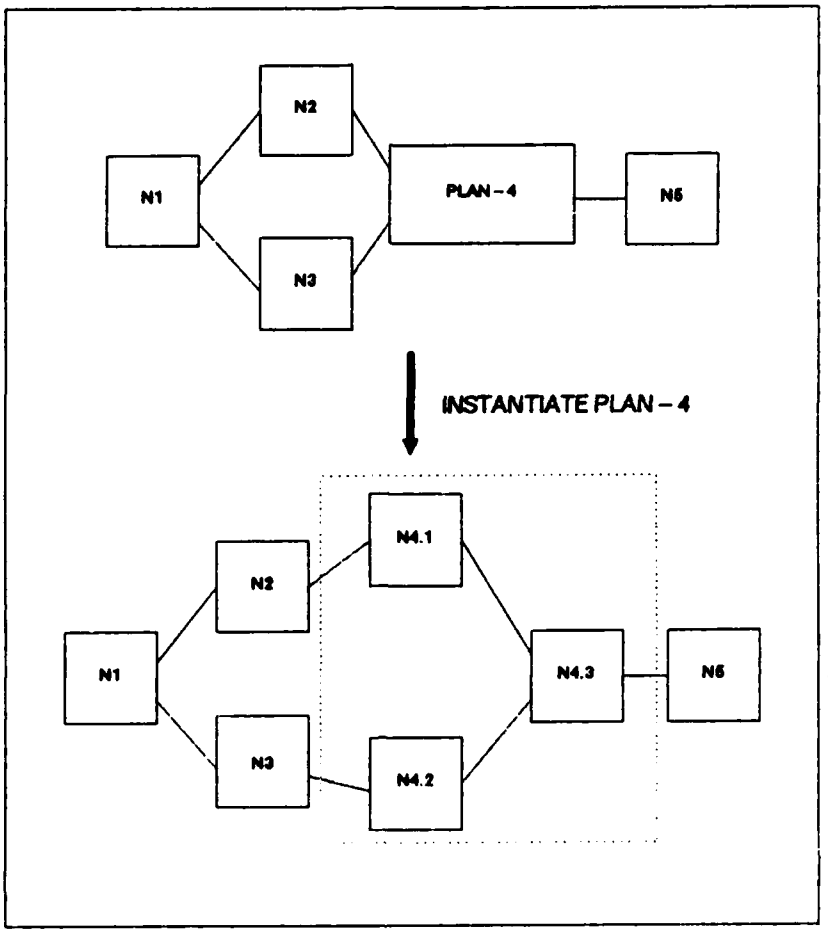


Figure 1.1 Plan Instantiation Example

Compilers have been developed specifically for dataflow computers that accept a language similar to ordinary procedural languages (Ackerman, 1982:15-16; Veen, 1981:130). There has been one effort to develop a compiled language specifically for the Evans and Sutherland PS300 (Yamaguchi, 1985:48-60). Although this may benefit the user who is familiar with the PS300, it would still require considerable effort for a new user to learn the language. An objective of the Graph Design Assistant is to allow a user to use the

PS300 without having to learn the syntax of a textual language.

Program Transformation Systems. Program transformation systems form a broad category of programming tools. "Transformational programming is a methodology of program construction by successive applications of transformation rules" (Partsch, 1983:201). The instantiation of a plan in the Graph Design Assistant can be viewed as applying a transformation to the functional network graph. In some of the program transformation systems the user selects transformations from libraries or catalogs (Partsch, 1983:206). This is similar to the selection of plans in GDA.

Programmer's Apprentice. The Programmer's Apprentice (PA) is a project developed at M.I.T.(Waters, 1985:1296). The PA can be classified as a program transformation system (Partsch, 1983: 224). It is included in a separately in this section because its approach is similar to that of the Graph Design Assistant. The Programmer's Apprentice "lies somewhere between language-oriented programming tools one hand, and automatic programming tools on the other" (Rich, 1978:444). The Programmer's Apprentice is designed to interactively aid and check the work of a programmer; it keeps track of details and frees the programmer to concentrate on the hard parts of the problem (Waters, 1985:1296).

Graph Grammars. There has been work in the theory of graph grammars which is relevant to the Graph Design Assistant. Graph grammar theory is uses dimensional graphs in the same way that Formal Language theory uses one dimensional strings (Ehrig, 1978:9). It has been suggested that graph grammars would be useful in the generation of programs, given that so many of the programming aids (flow charts, structure charts, block diagrams, etc.) are in the form of graphs (Nagel, 1979:71). In the case of dataflow programming languages, graph grammars are particularly appropriate because dataflow programs are easily represented with graphs (Davis, 1982:26).

A System for Interactive Design (SID) is an example of a system which uses a graph grammar as its basis (Kunii, 1980:33). SID has been used to design hospital information systems (Kunii, 1980:33) and petrochemical plants (Buchmann, 1979: 732).

In the design of the GDA, an attempt was made to define the grammar of the PS300's dataflow language as a separate part of the system. The objective of the separation of the grammar of the FGN language is that the GDA system could be used for other purposes by replacing the grammar definition.

Visual Programming. A recent term which has been used to describe some programming systems is "visual programming" (Jacob, 1985:51). The idea is to construct programs by creating and editing diagrams rather than text. Others have

stated that since a dataflow programs can be viewed as a graph, it would be advantageous to do away with a text representation and directly manipulate the graph (Davis, 1982:27).

The Graph Design Assistant presents the dataflow program to the user in the form of a graph. The user creates and edits programs by making changes directly to the graph.

1.5 Sequence of Presentation

Chapter II presents the requirements definition by examining the needs for a robot simulation facility, and for the Graph Design Assistant. The system architecture and overall design of the GDA is presented in chapter III. In addition the development strategy for GDA is described and justified. Chapter IV discusses the detail design and implementation of the Graph Design Assistant. Finally, the results, conclusions and recommendations are presented in chapter VI.

II. Requirements and High-Level Design

The first step in the design of the Graph Design Assistant was to define the requirements for the system. Requirements are a description of a system to be build (DeMarco, 1979:412).

The first section in this chapter is a description of the general requirements for the robot simulation facility. This provided the context for the Graph Design Assistant. Next, the Functional Graph Network (FGN) language is examined. FGN is the desired output of the Graph Design Assistant; therefore, it is a major factor in the determination of the performance requirements for GDA. Next, the specific requirements for the Graph Design Assistant are outlined with a series of design goals. Finally, an example scenerio is used to make explicit the interactive user requirements of GDA.

2.1 General Requirements for a Robot Simulation Facility

The requirements for the robot simulation facility were driven by the anticipated users of the system. Three different "user roles" were defined. Each role corresponds to a specific purpose for using the simulation facility. The three roles are Robot Display Designer, Robot Control

Program Developer, and Robot Model User. Figure 2.1 is a diagram which shows the three different user roles, the processes which comprise the system, and the output from the processes.

Robot Display Designer. One function the robot simulation facility must perform is to allow for the creation of robot models. As shown in Figure 2.1 there are two processes involved in the creation of a robot simulation model. They are the Display Tree Design Tool and the Graph Design Assistant.

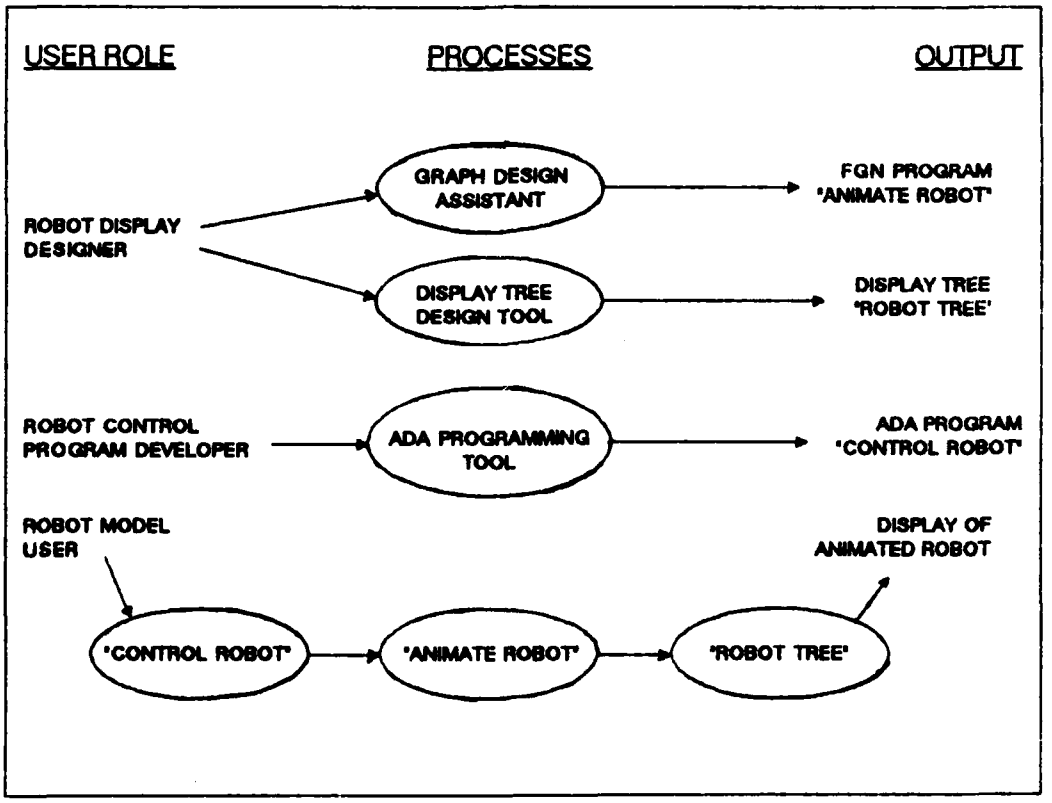


Figure 2.1 User Roles in the Robot Simulation Facility

The first process used by the Robot Display Designer is Display Tree Design Tool. The processes allows the user to design and edit display trees. A display tree is a data structure which contains the three dimensional coordinate information which defines the shape of the image. When the display tree is loaded into the PS300, the image is displayed on the PS300's screen. In the example shown in Figure 2.1 the display tree is named "Robot Tree"; it describes the shape of a robot.

The second process used by the Robot Display Designer is the Graph Design Assistant. With the GDA, the Robot Display Designer creates a functional graph network, named "Animate Robot" in the example in Figure 2.1, which controls the movements of Robot Tree.

The two products created by the Robot Display Designer, together define the animated robot model. When loaded into the Evans and Sutherland PS300, the display tree defines a three dimensional image of a robot, and the functional network controls the movement of the robot.

In the example shown in Figure 2.1, the robot model is designed to interface to a robot control program running on the host computer connected to the PS300. It should be noted, however, that it is possible to design robot models that are controlled directly through input devices on the PS300. The assumption in the example is that the users of

the robot simulation facility have an interest in developing robot planning and control programs.

Robot Control Program Developer. Another role that a user of the robot simulation facility might assume is that of a Robot Control Program Developer. In Figure 2.1, the Robot Control Program Developer is using an Ada programming tool to develop a program named Control Robot. (The program language Ada was choosen as an example, in fact, other programming languages could be used its place.) In Figure 2.1 the robot simulation model created by the Robot Display Designer would be used to test the program "Control Robot".

Robot Model User. The last role defined for the users of the robot simulation facility is that of the Robot Model User. In this role the user gives high level commands to "Control Robot". The process "Control Robot" gives lower level commands to "Animate Robot" to cause the simulated robot's movements.

As can be seen in Figure 2.1 the Graph Design Assistant is part of the process used by the Robot Display Designer. The purpose of GDA is to create a functional network which controls the movements of the animated robot. The functional network is created using a special-purpose dataflow language named the Functional Graph Network (FGN) language. The next section describes this language.

2.2 Functional Graph Network Language

The Functional Graph Network (FGN) language is the special-purpose dataflow language which is used to program the Evans and Sutherland PS300 graph workstation (Davis, 1984). A FGN program consists of a series of statements which are instructions to the PS300 to construct functional networks and display trees within the PS300's memory. Once a network and display tree are in memory, the PS300 scans the display tree and shows the resulting image on the display screen (Evans and Sutherland, 1984a). The functional network can have connections to the PS300's input devices (tablet, dials, mouse, and function keys), and can output data to the display tree. In this way, the functional network allows the user to manipulate the image with the input devices. Additionally, a host computer connected to the PS300 can send data to the functional network and manipulate the image.

In this section, a simple example of a display tree, a functional network, and the FGN program which generates the tree and network, is developed. Figure 2.2 illustrates the functional capabilities of the "Rotate Squares" example. Within each of the rounded squares is an image that would appear on the display screen of the PS300. The left most screen is the initial image of two concentric squares.

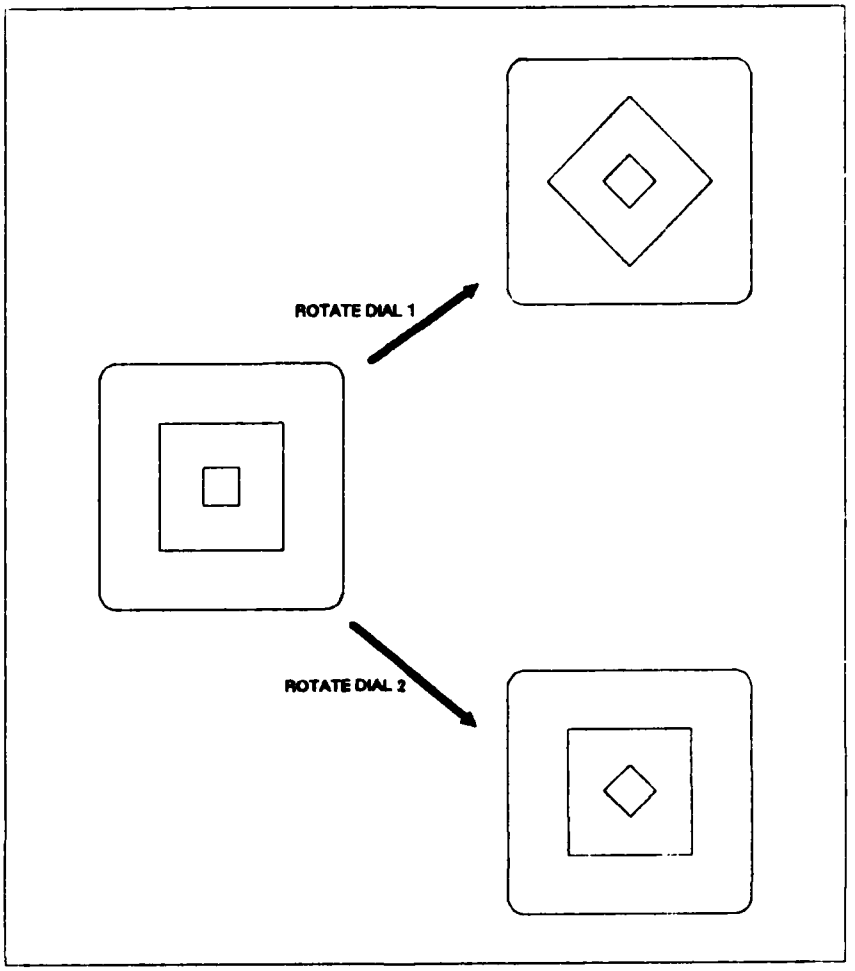


Figure 2.2 Rotate Squares Example: Functional Capabilities

Turning dial 1 causes both squares to rotate, while turning dial 2 causes the inner square to rotate relative to the outer square. The following section explains how the the display tree and the functional network for the "Rotate Squares" example are constructed.

2.2.1 Display Trees

A display tree is a hierarchical structure within the Evans and Sutherland PS300. The display tree consists of elements linked together in a tree like structure. The elements are vector lists, sub-tree structures, and transformation nodes. The PS300 displays an image by constantly scanning a display tree, applying transformations in the nodes to vector lists and displaying the resultant vectors on the screen.

Figure 2.3 is a diagram which represents the display tree in the "Rotate Squares" example. Starting at the top of the tree, the triangle is a sub-tree structure named "double-squares". The node labeled "rot_out" (labeled nodes are shown as two concentric circles) is a transformation node used to rotate the outer and inner squares. The unlabeled node (unlabeled nodes are shown as single circles) is a transformation node used to scale the inner square. The node labeled "rot_in" is a transformation node used to rotate the inner square. The box labeled "square" is a list of vectors or lines (stored as the end points of each line) that define the 4 sides of a square.

In the display tree in Figure 2.3 there are two "instances" of the square depicted as lines from "rot_out" and "rot_in". The "rot_out" transformation node is designed so that when a rotation transformation is applied to the

node, both instances of the square (the inner and outer squares) rotate together. The unlabeled transformation node directly below "rot_out" contains a scaling transformation which makes the inner square smaller than the outer square. The "rot_in" transformation node is designed so that when a rotation transformation is applied to the node, only the inner square rotates.

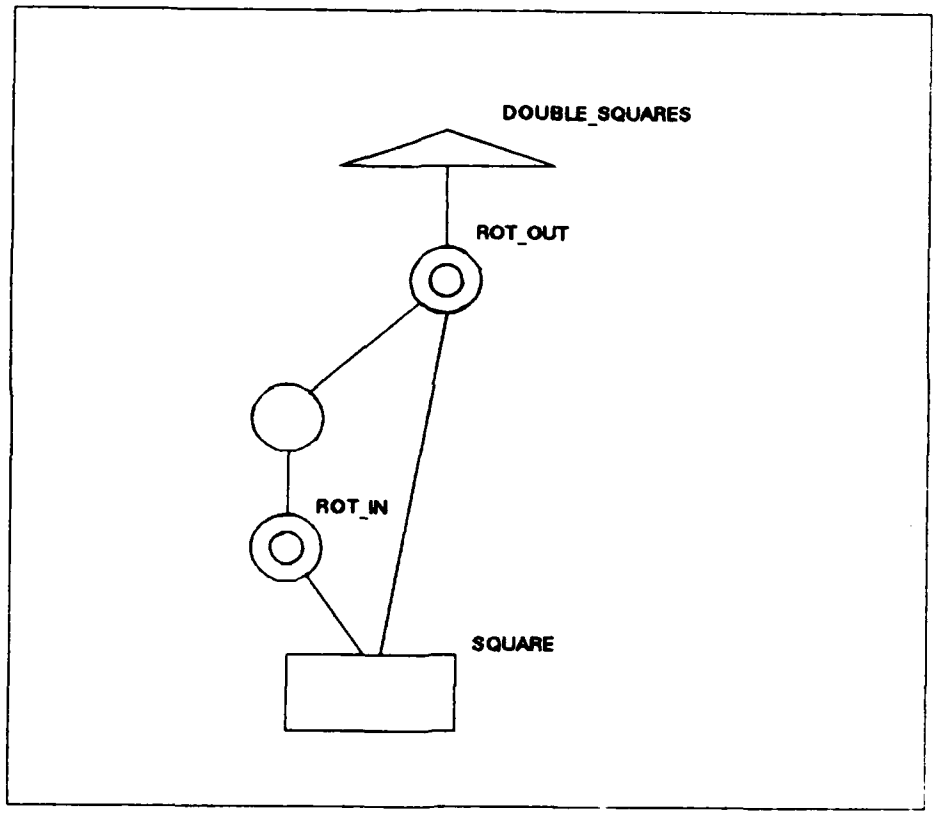


Figure 2.3 Rotate Squares Example: Display Tree

The image defined by the display tree in Figure 2.3 can manipulated by making connections from a functional network to the labeled nodes. The one transformation node is

unlabeled because the scaling transformation was designed to remain constant; therefore, it does not need a label as a connection point to the functional network. The next section describes functional networks and how they relate to display trees.

2.2.2 Functional Networks

The PS300 is programmed by constructing functional networks that reside in the PS300's internal memory. The functional networks of the PS300 can be viewed a dataflow graph (Davis, 1984:1.2). In a dataflow graph, the nodes of the graph are connected to each other with arcs. Data tokens flow between the nodes on the arcs. Each node performs a function on its input data tokens and produces output data tokens.

There are many types of functional nodes used in the PS300. Many of the nodes perform functions designed specifically for a graphic operations. For example, there are functions which support matrix and vector manipulations. These are useful for the transformation of three dimensional coordinate data. Other functions are more general purpose in nature, allowing the construction of general purpose computational structures. For example there are functional nodes which are useful in building iterative loops (Davis, 1984:2.13).

Several types of data tokens which flow between the functional nodes in the network. The data tokens can be simple or composite date items. The simple data items can be integers, reals, or boolean values. The composite data items can be matrices or vectors.

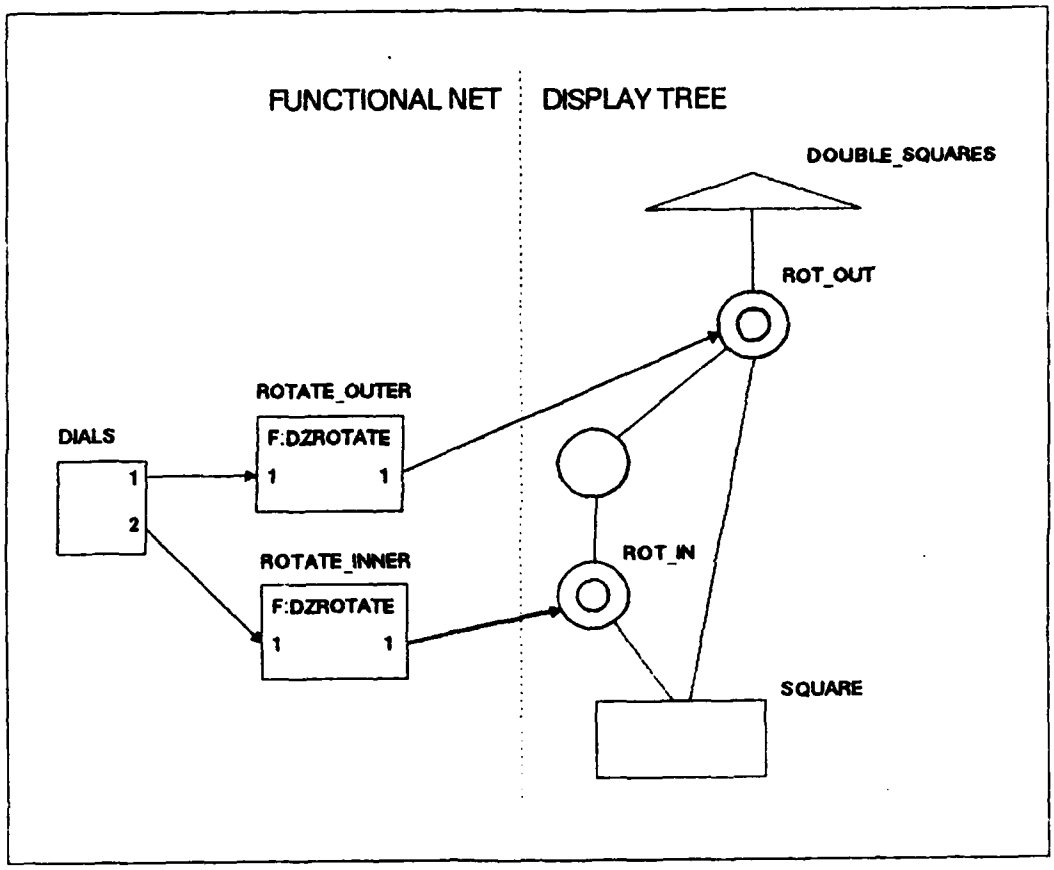


Figure 2.4 Rotate Squares Example: Functional Network

The functional network for the "Rotate Squares" example is shown in Figure 2.4 on the left side. There are three functional nodes (shown as boxes), labeled "dials", "rotate_outer", and "rotate_inner". The "dials" functional

node is used to get input from the PS300's dial inputs (Evans and Sutherland, 1984b:5). The PS300 has a set of dials that the user can turn. If the output of a dial has been connected to a functional network, then when the user turns the dial, a stream of real numeric data tokens are sent out from the functional node. In this example, dials 1 and 2 are used as inputs to the functional network. The "rotate_outer" and "rotate_inner" nodes are F:DZROTATE type nodes. Every time a F:DZROTATE node receive a numeric data token on its input, it applies a rotation about the Z axis to an internal transformation matrix. This matrix is then sent out on output 1 of the F:DZROTATE node. Since the output of "rotate_outer" and "rotate_inner" are connected to nodes in the display tree, each time dial 1 or 2 is turned the transformation matrix in the nodes "rot_out" or "rot_in" are replaced to reflect an incremental rotation. The final effect is to rotate the outer and inner squares when dial 1 is turned, and to rotate the inner square relative to the outer square when dial 2 is turned.

2.2.3 FGN code

In the description of the Functional Graph Network language, the programs have been described in graphical terms. In fact, the graph structures are specified to the PS300 using a textual program, which looks like an ordinary

programming language. Below is the code which builds the display tree and functional network used in the "Rotate Squares" example.

```
1.   square := vector list
                .5,.5 .5,-.5 -.5,-.5 -.5,.5 .5,.5;
2.   double_squares := begin_structure
3.        rot_out := rotate 0;
4.        instance square;
5.        scale .4,.4;
6.        rot_in := rotate 0;
7.        instance square;
8.   end_structure;

9.   rotate_outer := f:dzrotate;
10.  rotate_inner := f:dzrotate;

11.  connect dials<1>:<1>rotate_outer;
12.  connect dials<2>:<1>rotate_inner;
13.  connect rotate_outer<1>:<1>double_square.rot_out;
14.  connect rotate_inner<1>:<1>double_square.rot_in;
15.  display double_squares;
```

The program begins with the specification of the display tree. Line 1 defines a vector list named "square", which is a unit square centered about the origin of the X,Y plane (the Z coordinates default to 0). Line 2 through line 8 define a structure named "double_squares". The two transformation nodes "rot_out" and "rot_in" are defined with in lines 3 and 6 with the "rotate" statement; the initial rotation is 0 degrees. The unlabeled transformation node used to scale down the smaller square is defined in line 5. Lines 4 and 7 establish the outer and the inner instances of a square.

The functional network for the "Rotate Squares" example is defined in lines 9 through 14. First the nodes "rotate_outer" and "rotate_inner" are created in lines 9 and 10. The node "dials" is a predefined system function node. In lines 11 through 14 the connections in the network and to the display tree are made.

The code to create the display tree and functional network is downloaded from a host computer to the PS300 workstation through a serial line. In line 15 the statement "display double_squares" causes the two concentric squares of the "Rotate Squares" example to appear on the screen.

The ultimate purpose of the Graph Design Assistant is to produce FGN code from a graphical specification as opposed to the textual program specification shown in the above example. With this goal in mind, the next step is to examine some design goals for the GDA.

2.3 Design Goals for the Graph Design Assistant

The purpose of the GDA is to allow the user to program the PS300 without having to learn the intricacies of its dataflow language. The design goals to accomplish this purpose follows.

Design goal 1: Visual programming system

The GDA allows the use to create a program by build a graph. The graph is presented to the user visually on the computer terminal. The user enters commands by pointing to items on the screen with a mouse. When an item has been "moused" it displays a menu of functions that can be applied to the item. The reason for the emphasis on visual programming is to free the user from having to know and remember a specific syntax (as is required when defining a functional network using the written text form of the FGN language).

Design goal 2: Create and use plans

A significant problem with programming the PS300 using the FGN language is that there is no method for defining a "macro". If a functional network contains several identical sub-graphs, the user has to write code to define each one individually.

The notion of a plan is similar to a macro. Once a plan has been defined, it can be instantiated (or called) to produce a graph structure. A plan is different from a macro, however, in that the plan is more flexible. A single plan may expand to different structures depending on the context.

Design goal 3: Design by selecting plans from a library

The advantage of being able to define and use plans is that a library of plans can be created. When the GDA is used in the robot simulation facility, a library of plans geared toward the creation of robot simulations will exist. This amounts to creating a higher level language which frees the user from the underlying details of the FGN language. Given a sufficiently extensive library of plans, the typical user will be able to construct robot simulations by selecting plans from the library.

Design goal 4: Consistency checking

In construction a functional network graph, there are many opportunities for a user to make mistakes. On of the goals for the Graph Design Assistant is to automatically detect errors during the construction of a graph and inform the user.

There are different types of consistency checks can make on a functional network. For example tests could be made for the following conditions: the inputs and outputs of each node need to be connected, output connections must be made to input connections that are compatible data types, and the functional network should be a "live" network.

"Liveness" is a property of a well defined behaved dataflow programs (Davis, 1984:3.2). A network is a alive if data tokens propagate throughout the graph. Data tokens must be available on all the input nodes of a functional node before the node fires. A dead network has nodes which can never fire.

2.4 User Interaction Scenario

The Graph Design Assistant is a "highly interactive" system. The following scenerio was devised in an attempt to define the interactive requirements for GDA.

The construction of FGN programs can be viewed as a part of a design process. In particular, the user of GDA is designing a graph which is a representation of the functional graph network. The model of the design process used in GDA is "abstract refinement" (Mostow, 1985:45). In the abstract refinement model, a design is constructed at a series of levels (as in Figure 2.5). The designer begins at the top level with a few abstract components. The designer refines components at lower levels. Each lower level corresponds to a refined design for a higher level component. (In conventional software engineering this method is known as top down structured design.) Although this method sounds simple, there can be difficulties if the components are not completely independent of each other. In

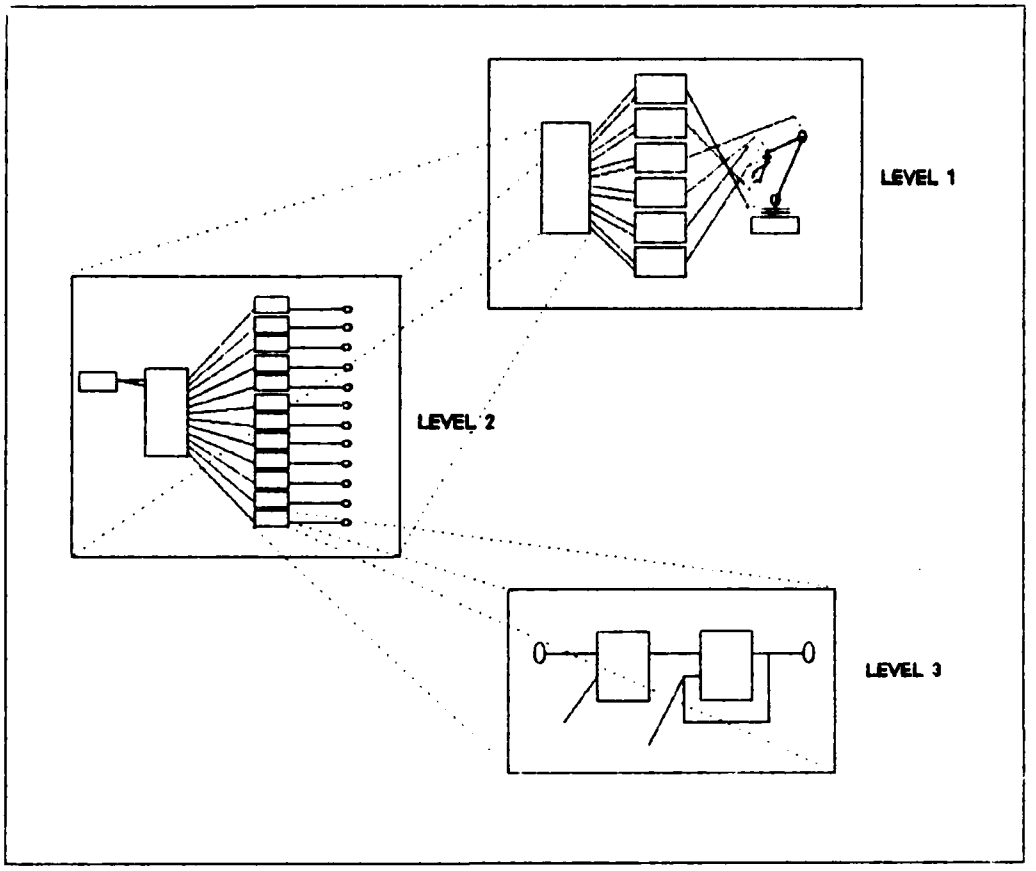


Figure 2.5 Scenario: An Example Design Hierarchy

the functional graph network, there are many interconnections between components; GDA checks the user's design for mistakes.

What follows is a description of a typical session using GDA to design a Functional Graph Network. The purpose of this desciption was to help define the functional specifications for the Graph Design Assistant.

The designer in this scenario designs a functional graph network to animate and control a 6 axes robot arm.

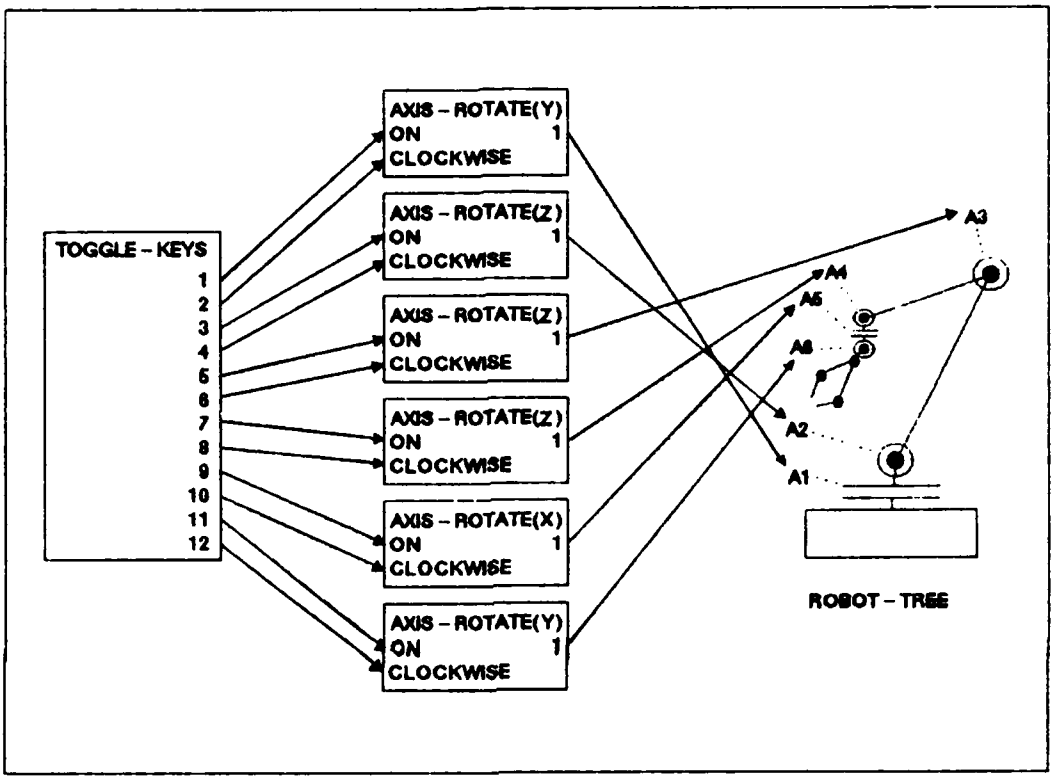


Figure 2.3 Scenario: Level 1 "Animate Robot"

The robot arm is controlled by 12 function keys on the keyboard of the PS300. Each axis of the robot is controlled with two keys. The first key starts and stops the rotation about an axis, while the second key changes the direction of the rotation.

Scenario: Design of level 1

The designer constructs a graph by instantiating nodes and making connections between nodes. The designer can either choose predefined nodes from a library or create new

sub-graph nodes. Figure 2.6 shows the design at level 1. The designer has instantiated six copies of the plan "rotate-axis(x/y/z)", and one copy of the display tree description "robot-tree". In addition, the designer has created a sub-graph node, "toggle-keys", which must be refined later.

The "robot-tree" node is a description of a display tree. In practice, this tree would be defined by some other development tool. The tool would generate the actual display tree, which is downloaded to the PS300, as well as a description of the display tree for use in the GDA. A display tree is a data structure that describes the image of the robot. It consists of vector lists and transformation nodes. The PS300 has a display processor which repeatedly scans the display tree, applying the rotation, translation, and scaling transformations to the vectors to produce the image on the display screen. The transformation nodes in the tree can be labeled so that they can receive input from the the function graph network.

The display tree description contains knowledge that GDA must have to build a functional network that will interface correctly with the display tree. The description contains a list of transformation nodes and information about constraints on the nodes. The robot arm described by "robot-tree" (Figure 2.6) has six joints. "Robot-tree" contains a list of the six nodes labeled "a1", "a2",..."a6".

The description also contains constraint information about each node; the joints are limited in the amount they can rotate and which axis (x, y, or z) they can rotate about.

The "toggle-keys" node is a sub-graph node. The detailed design of this node is described later. (The system automatically puts a notice on the agenda that design of "toggle-keys" must be refined. The designer selects the notice from the agenda when he decides to complete the design.) The "toggle-keys" sub-graph allows the function keys to act as boolean toggle switches. A boolean toggle switch alternately outputs a "true" then a "false" each time the switch is pressed.

The "axis-rotate(x/y/z)" node is a plan that generates an network which causes a transformation node on a display node to rotate about the x, y, or z axis. The inputs to this node are boolean values. If the "on" input is true, then the node outputs rotational transformations until the input changes to false. If the "clockwise" input is true, the rotation is clockwise, otherwise it is counter clockwise. When FGN instantiates this plan, it automatically checks the description of the "robot-tree" to determine which axis to rotate, and what rotation limits to build into the network.

In the final step of the design of level 1, the designer selects the "toggle-keys" node and tells GDA to "zoom in" on this node. The window displaying the current

graph clears and the designer begins the design of the sub-graph for "toggle-keys".

Scenario: Refinement of level 2's "toggle-keys"

The design process at level 2 of the hierarchy is exactly the same as for level 1. The designer selects predefined nodes from the library, or creates new sub-graph nodes (to be refined later). Figure 2.7 shows the final design of "toggle-keys". The 12 identical "toggle" nodes could have easily been predefined nodes selected from a library, however for the purposes of this example the designer created them as sub-graph nodes. The "fkeys" and "f:croute" nodes are primitives of the FGN language. The "fkeys" node outputs an identifying integer each time a function key on the PS300's keyboard is depressed. The "f:croute" function node has two inputs. Input 1 selects which output the data from input 2 is directed to. In this configuration, whenever function key 6 is depressed, a 6 is sent from output 6 of "f:croute". This in turn causes the "toggle" node (connected to output 6 of "f:croute") to toggle it's output from true to false, or false to true.

Next, the designer selects one of the "toggle" nodes to refine the design (since all 12 of the toggle nodes are identical, the user only has to design "toggle" once).

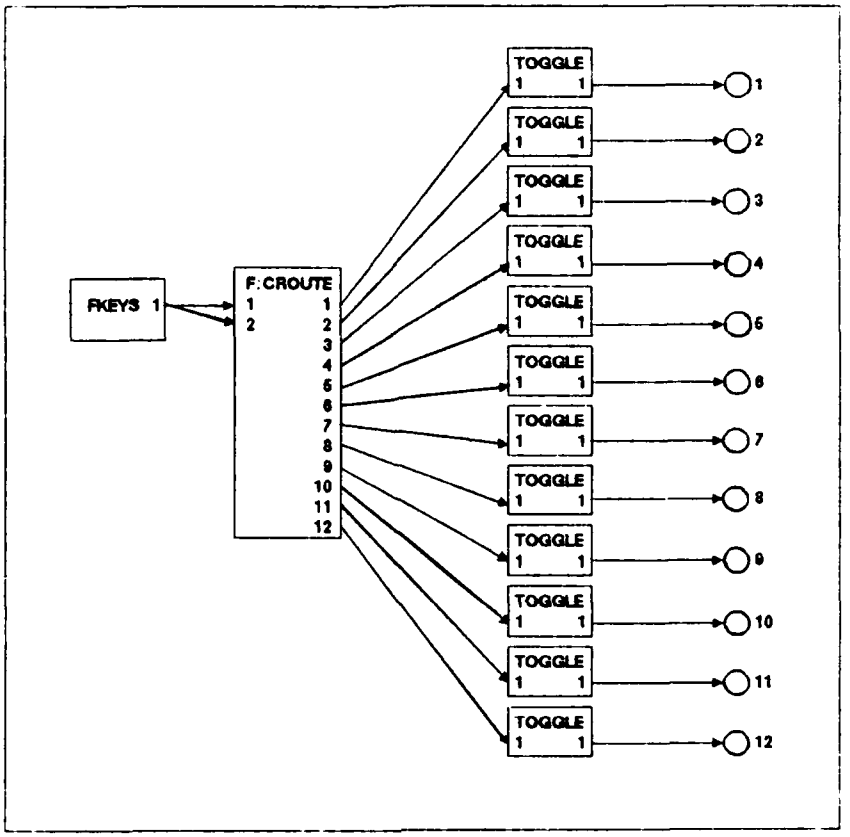


Figure 2.7 Scenario: Level 2 "Toggle Keys"

Scenario: Refinement of level 3's "toggle"

Since this is the lowest level of the design, the user instantiates primitive FGN function nodes. The toggle function is shown in Figure 2.8. The "f:constant" node works by triggering whenever it receives data on input 1. When ever "f:constant" triggers it outputs the constant value on input 2. In this case "f:constant" outputs a true whenever it receives data on input 1. The output from

"f:constant" goes to the "f:xor" node. This performs an "exclusive or" operation on the input 1 (always true) and input 2 (the last output value). The effect is that the "f:xor" node outputs a data token that is the compliment of the node's previous output.

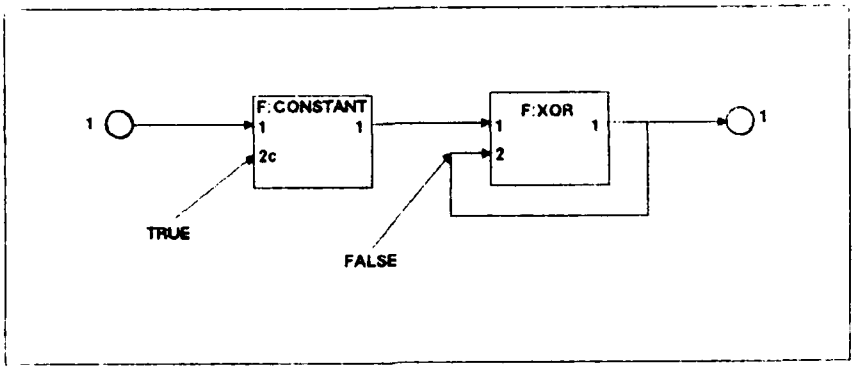


Figure 2.8 Scenario: Level 3 "Toggle"

The above scenario describe some of the possible interactions between the user and GDA. Two aspects of GDA, however, were not brought out in the scenario. One is the means of interaction between the user and GDA. GDA uses graphic to display the state of the design. The user uses a mouse to point to elements on the display to carry out his commands. For example, if the user wants to connect two nodes, he points to the output connection of the source node, clicks the mouse, then points to the input connection of the destination node, and again clicks the mouse to complete the connection. A line is then drawn by GDA on the screen between the two nodes. The second aspect of GDA not

brought out in the scenario, is the constant consistency checking that occurs during the design process. Suppose that in the above connection example, the output and input nodes had been incompatible. Maybe the output produced an integer token while the input only accepted real tokens. GDA would detect the conflict and notify the user of the error. In addition, it would post a notice on the agenda to insure that the user would eventually correct the error.

The above scenerio described the creation of a functional network using the Graph Design Assistant. The scenario served as a functional description and was used as a basis to begin the design presented in the next chapter.

III. Design

The design process of the Graph Design Assistant did not follow conventional software engineering practice. Because of this, the first section of this chapter examines and justifies the development strategy used in this thesis effort. Next, the design for GDA's system architecture is described. And finally the overall software design is examined.

3.1 Development strategy

One way to categorize software development projects is that they are either "design problems" or they are "implementation problems" (Sheil, 1983:20). Because the requirements were not well defined, it was felt that the Graph Design Assistant fell into the category of a "design problem".

In a project that is an "implementation problem", the requirements are well understood from the start. It may be that the project is a redesign of and existing system or design of a system which is similar to an existing system. This is not to say that the "implementation problems" are trivial; the size of many projects can make them very difficult. There is, however, a well developed body of

software engineering knowledge that can aid in the development of large "implementation problems".

The "waterfall" model of lifecycle development is useful in dealing with "implementation problems". The software lifecycle model is a step by step approach, which begins with requirements definition, design, code and debugging, testing, and finally operations and maintenance (Boehm, 1976:72). Although there may be some feedback in the development process, steps may be repeated, the general trend is to finish one step before proceeding to the next. It is very important, therefore, in a lifecycle development that the requirements are well defined before proceeding to the development phase.

Unfortunately, in a "design problem" type project, the requirements are often difficult to define until the system has been built (Floyd, 1984:2; Sheil, 1983:20). The requirements for the system were poorly defined at the beginning of the effort. There were no available "dataflow programming assistant" systems to model GDA after. In an attempt to alleviate the problems with imprecise specifications, two related strategies, "prototyping" (Floyd, 1984) and "exploratory programming" (Sheil, 1983), were used in the development of the Graph Design Assistant.

3.1.1 Prototyping

The definition of prototyping in relation to software development is a problem. The literal meaning of prototype, "first of a type", and its use in other engineering disciplines does not correspond with the way it is used in software engineering. For the purposes of this effort, a prototype is an "experimental prototype" primarily designed to enhance the specification of a system (Floyd, 1984:9).

It should be mentioned that prototyping is not completely incompatible with the lifecycle approach to software engineering. First, the development of a prototype can be viewed as the first step (requirements definition) in the normal lifecycle. Second, the process of developing the prototype may undergo the phases of the lifecycle, although the steps might not be rigidly controlled.

The success of a prototyping strategy depends in a large part upon the availability of appropriate development tools. Because of the possibility that the prototype may serve as a "throwaway" learning device, it is important that the prototype can be developed at a relatively low cost (Floyd, 1984: 10). For this reason, "exploratory programming" is the second part of GDA's development strategy.

3.1.2 Exploratory Programming

Exploratory programming has been defined as the "conscious intertwining of system design and implementation" using advanced programming environments (Sheil, 1983:19). The second strategy in the development of the Graph Design Assistant was to use tools developed for Artificial Intelligence (AI) research and applications. Researchers in AI have attempted to build large systems in order to solve poorly understood problems. Usually they are built quickly with small programming teams. During the course of development the programs undergo many modifications due the fact that the problems are so difficult. A consequence of the challenges of AI research have been the development of powerful "exploratory programming environments" (Sheil, 1983:22).

Early in the design of the Graph Design Assistant the decision was made to such a tool developed for AI applications. The Knowledge Engineering Environment or KEE (Intellicorp, 1985c). KEE is a hybrid development environment that combines frame-based knowledge representation, object oriented programming, and rule-based reasoning (Fikes, 1985:906; Kunz, 1984: 41).

The version of KEE which was available ran on a Symbolics 3600 "lisp machine" (Symbolics, 1985a). A feature of KEE that proved extremely useful is that it allows easy

access to the underlying Symbolics development environment. The Symbolics supports Zetalisp as its programming language and operating system (Symbolics, 1985b). The Zetalisp environment provides many programming tools such a syntax sensitive editor, symbolic debugger (Symbolics, 1985c). Most importantly for the implementation of the Graph Design Assistant, the Zetalisp environment has many functions for doing graphic and windows (Symbolics, 1985d:73-126), and building mouse and menu based user interfaces (Symbolics, 1985d: 207-255)

3.2 Graph Design Assistant Architecture

The decision to use KEE in the implementation of the Graph Design Assistant had a large influence in the design of the system architecture. In this section, the system architecture is examined. First, the hardware configuration of the robot simulation facility is shown. Then, the software architecture of the GDA is described.

3.2.1 Robot Simulation Facility Hardware Configuration

The hardware used in the present configuration of the robot simulation facility consists of a Symbolics 3600 computer, a VAX 11/780 computer, and an Evans and Sutherland PS300 graphic workstation.

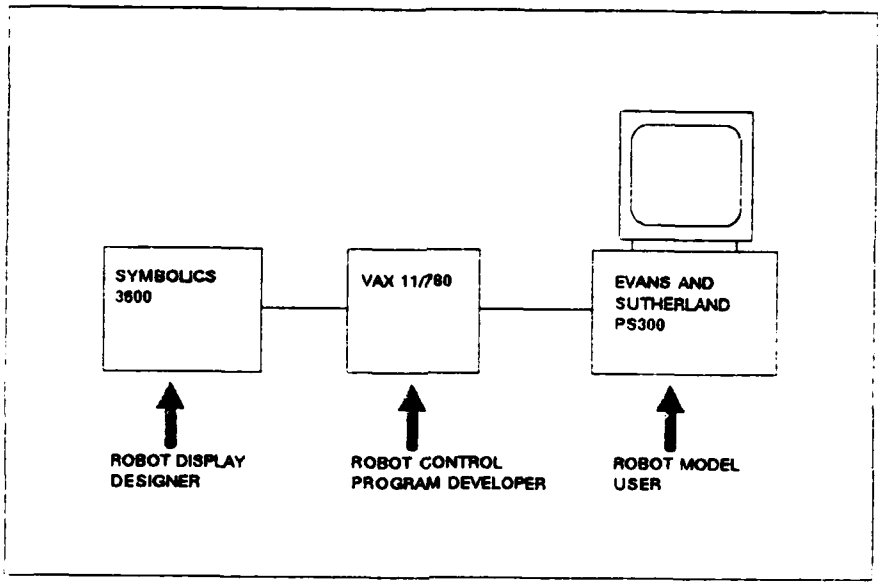


Figure 3.1 Robot Simulation Facility Hardware Configuration

Figure 3.1 shows the components and their connections. At the present time a serial line connects the Symbolics to the VAX (an Ethernet connection is planned) and the VAX to the PS300.

At the bottom of Figure 3.1, the relationship between the hardware components and the user roles (defined in chapter 2) are shown. The Robot Display Designer develops FGN programs on the Symbolics using the Graph Designer Assistant. The programs are downloaded to the PS300 through the VAX. The Robot Control Program Developer write his control programs on the VAX. The Robot Model User interacts with a model through the PS300 workstation.

A crucial component of the robot simulation facility is the Evans and Sutherland PS300 workstation. This decision

to uses this device was made because it is well suited for displaying animated robot simulations. The PS300 is capable of displaying three dimensional images. The images can be animated in real-time. The calculations to do the rotations, scalings and translations used in the animation of the image are made with special purpose hardware residing in the PS300 (Evans and Sutherland Computer Corporation; 1984a; Foley, 1982:418-421)

The VAX 11/780 acts as a host computer for the PS300 workstation. The PS300 can emulate a normal terminal as well as performing its graphic functions. Because the PS300 performs its graphic functions in local hardware, it makes few demands on the VAX (it is likely the the 11/780 will be replaced with a smaller model of the VAX). The VAX can serve, however, in the development and execution of planning and control program which will interact with simulations running on the PS300.

The decision to include the Symbolics 3600 in the robot simulation facility was based on the availability of the powerful, AI development tools available within the Zetalisp environment. Since this first implementation of the Graph Design Assistant is a prototype, a decision might be made to implement future versions of GDA on the VAX host computer.

3.2.2 Graph Design Assistant Software Architecture

The organization of the GDA was strongly influenced by the choice to implement the system with KEE. In addition, the method for describing the system was affected by KEE.

It is difficult to describe the Graph Design Assistant using documentation techniques such as a data dictionaries and structure charts. A system implemented in a conventional programming language can be defined by the structure of the system's data and the procedural code that operates on the data. Normally the program is conceived as a hierarchy of procedures or modules, and the software architecture can be explained in terms of that hierarchy (structure charts for example).

Unfortunately, a system implemented in KEE is difficult to describe using standard software engineering graphical aids. The primary component of a KEE implemented system is a "knowledge-base". KEE's knowledge-bases combine both data and procedures in a single unit, which makes it difficult to describe the system with data dictionaries and structure charts.

The approach that has been taken to describe the Graph Design Assistant is to first describe the in general, knowledge-bases and their and their relationships (in this chapter). Then the objects which comprise the knowledge-bases are described in detail (in the next chapter).

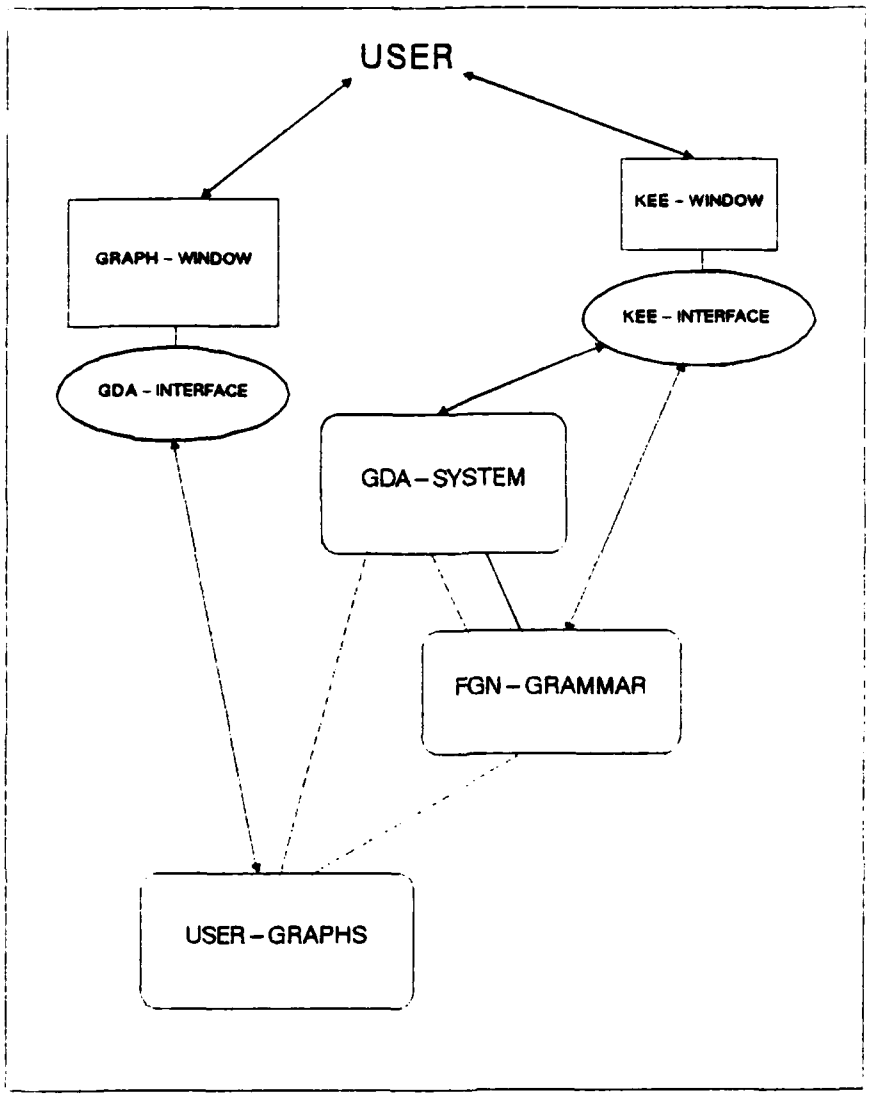


Figure 3.2 Graph Design Assistant Architecture

GDA knowledge-bases. Figure 3.2 shows the knowledge-bases of the Graph Design Assistant and indicates the relationships between them. In addition to the knowledge-bases, the processes which comprise the user interface are shown.

The knowledge-bases are depicted in Figure 3.2 as boxes with rounded corners. They are GDA-SYSTEM, FGN-GRAMMAR, and USER-GRAPHS. A knowledge-base contains either information which describes the attributes of objects, or the objects themselves. The knowledge representation scheme used by KEE represents objects as a taxonomy of "frames" (Fikes, 1985: 907).

Frames have any number of slots. The slots may contain declarative or procedural knowledge. The frames are arranged in a tree structure of object classes. A class is the description of an object as opposed to the actual object (members). A frame inherits traits (slots) from its parent classes. The top of the hierarchical is the most general description of a class of objects. Lower level frames refine the description of objects, while the "leaves" of the tree structure are members of the class.

For example in Figure 3.2 the knowledge-base GDA-SYSTEM contains the most general description of objects. The solid line from GDA-SYSTEM to FGN-GRAMMAR denotes that there are frames in FGN-GRAMMAR with are child classes of frames in GDA-SYSTEM. In other words, GDA-SYSTEM describes "generic" graph elements, while FGN-GRAMMAR describes graph elements that are specific to the Functional Graph Network language of the PS300. The dashed line from GDA-SYSTEM to FGN-GRAMMAR, indicates that some of the objects are members of a parent class defined in GDA-SYSTEM.

The FGN-PLANS knowledge-base contains the members of the plan library. Plans are structures similar to macros which can be expanded into complex structures to use in the building of graphs.

The USER-GRAPHS knowledge base contains only members of the classes defined in the other knowledge-bases. This knowledge-base contains the functional network graphs that the user is currently creating and editing.

The elliptical shapes in Figure 3.2 indicate "window processes". Window processes are attached to user interaction windows (indicated a rectangles). Windows are displayed on the screen of the Symbolics 3600. The window processes control the interaction with the user through the mouse or keyboard on the Symbolics 3600.

The process labeled KEE-INTERFACE was not as part of the Graph Design Assistant development, but is a built part of the KEE system. It is shown in Figure 3.2 to indicate that the knowledge-bases can be modified directly from KEE.

The process labeled GDA-INTERFACE was, on the other hand, constructed during the development of GDA. In a sense this part was developed outside of the KEE environment because certain graphic functions were needed which were not available within KEE. The GDA-INTERFACE controls the creation and editing of graphs within the USER-GRAPHS knowledge-base. The user the mouse on the graph window to

call up menus and execute commands which update the USER-GRAPHS knowledge-base.

The lines between the three knowledge-bases in Figure 3.2 are indications of relationships between frames within the respective knowledge-bases. The solid line indicates a "sub-class" relationship and a dashed line indicates a "member" relationship. A particular object is described by a hierarchy of classes; the classes are most general at the top of the tree, and become more specific towards the bottom of the tree. At the very bottom of a tree, the object itself is a member of a class.

Figure 3.3 is a closer look at the relationships between the three knowledge bases in the Graph Design Assistant. It shows, for illustration purposes, a few example frames within each of the knowledge bases.

The object labeled DZR_12 within the USER-GRAPHS knowledge base in an part of a functional network graph. It is described by the class DZROTATE, which is in turn a sub-class of PRIMITIVE, which is a sub-class of GRAPH-ELEMENT. The class DZROTATE is specific to the PS300's FGN language, hence it resides in the FGN-GRAMMAR. The class PRIMITIVE, on the other hand, describes an element that could be in many different dataflow grammars. In other words, it is a

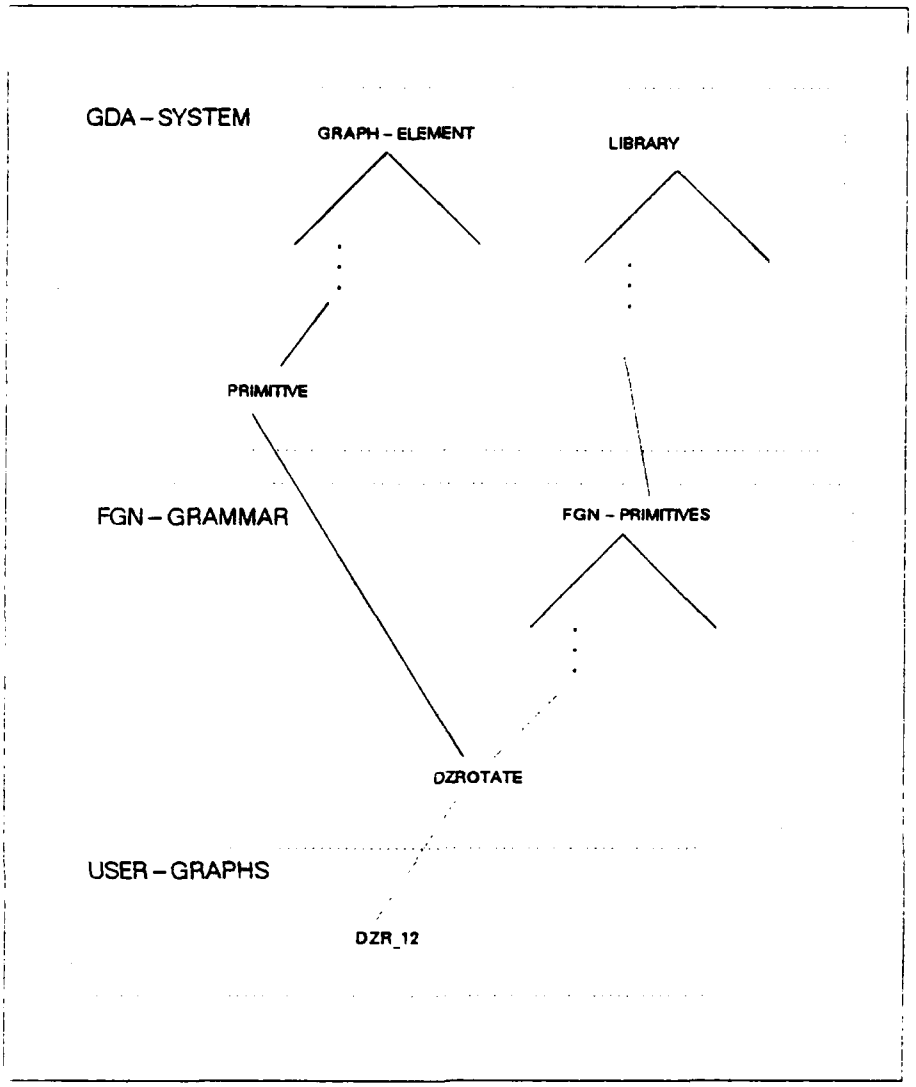


Figure 3.3 Graph Design Assistant Knowledge Bases

more general class. The class GRAPH-ELEMENT is even more general; it consists of all the elements which go to make up a graph (including primitives).

The DZROTATE frame shown in Figure 3.3, in addition to being a class description, is also a member of the GDA's library. It's a sub-class of FGN-PRIMITIVE (a FGN specific

class), which is a sub-class of LIBRARY (a general class applicable to any dataflow grammar).

In this chapter the strategy to develop a prototype using an "exploratory programming" environment was described. The high-level design of the Graph Design Assistant was discussed. In the next chapter, the design and implementation details of GDA are presented.

IV. Detailed Design and Implementation

The majority of the Graph Design Assistant system was implemented using the Knowledge Engineering Environment (KEE) (Intellicorp, 1985). A system is created with KEE by defining knowledge bases. Knowledge bases contain both the procedural and declarative knowledge about the system.

The chapter begins with a description of the representation of dataflow graphs. Next, some of the knowledge bases the Graph Design Assistant are described. Finally, the detail design and implementation of GDA's user interface are presented.

4.1 Representation of Dataflow Graphs

In the previous chapter, the "abstract refinement" model of the design process was discussed. Because of the design process, the user view of the dataflow graph is a hierarchical tree structure. At the top of the tree are are high level representations of graphs, while at the bottom of the tree are primitive functional nodes.

Figure 4.1 shows the different design levels of a dataflow graph as conceived by a user. The design of the overall graph G1 is shown at level 1. It consists of two

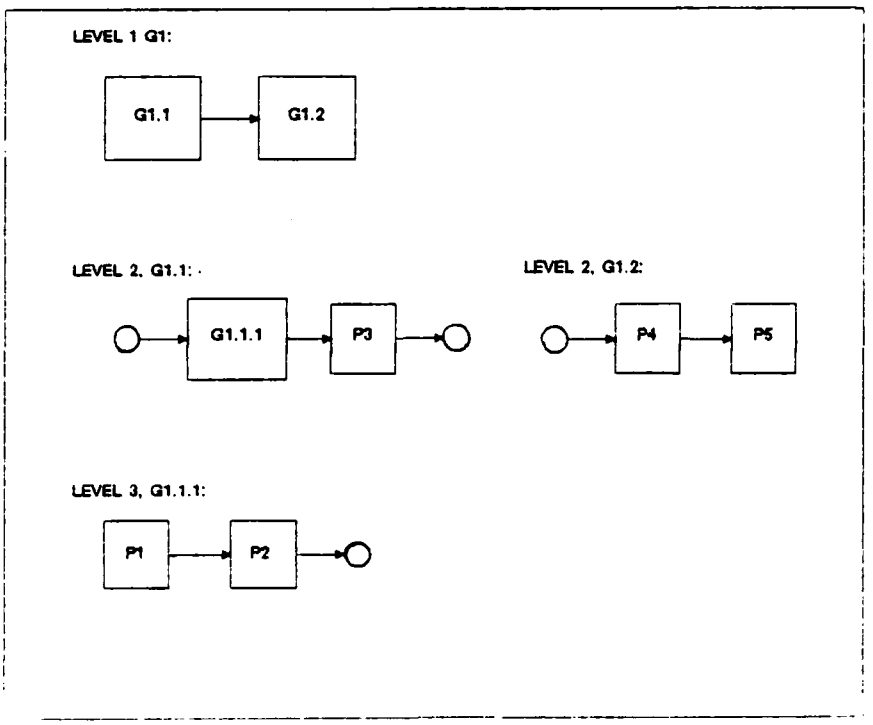


Figure 4.1 User's Design Hierarchy

sub-graphs, G1.1 and G1.2. The the output from graph G1.1 is directed to the input of G1.2.

Level 2 of Figure 4.1 shows further refinement of the design. The graph G1.1 consists of an input connector, a sub-graph G1.1.1, a primitive P3, and an output connector. The graph G1.2, also at level 2, has an input connection, and two primitives, P4 and P5. Level 3 of Figure 4.1 is a refinement of the graph G1.1.1. It contains two primitives, P1 and P2.

The functional network graph which is produced by the hierarchical design of Figure 4.1 is actually a "flat" structure. The hierarchical structure is due to the user's

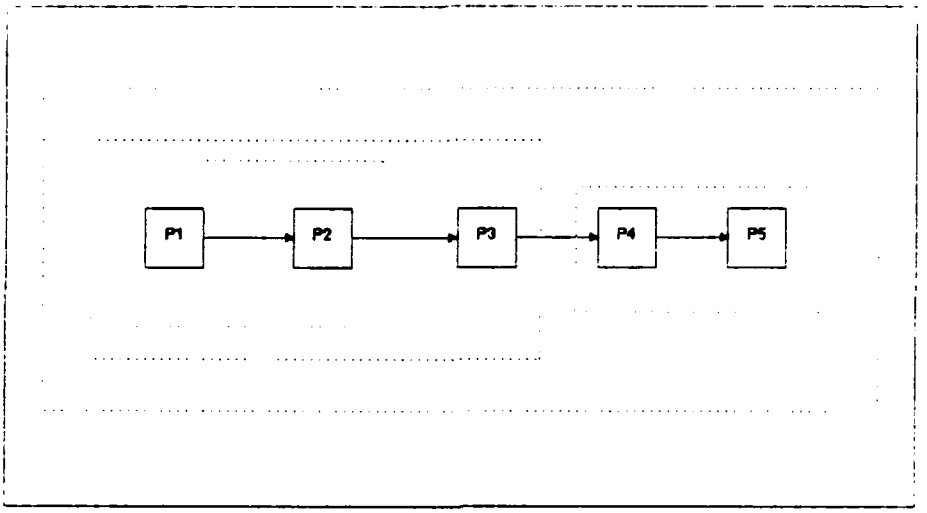


Figure 4.2 "Flat" Functional Network

design abstractions in the form of sub-graphs. Figure 4.2 shows the dataflow graph which consists of the primitives P1, P2, P3, P4 and P5. The light "dotted" square outlines show the sub-graphs that produced each part of the network.

In the designing GDA, a representation for dataflow graphs was devised that captured the hierarchical design information, and yet was easily transformed into a flat functional network.

A simple two level design hierarchy was used to explain the representation of functional networks. Figure 4.3 shows the user's view of the simple functional network. The rectangles represent the image the user would see in the Graph Window. The top level graph G1 (the label runs along

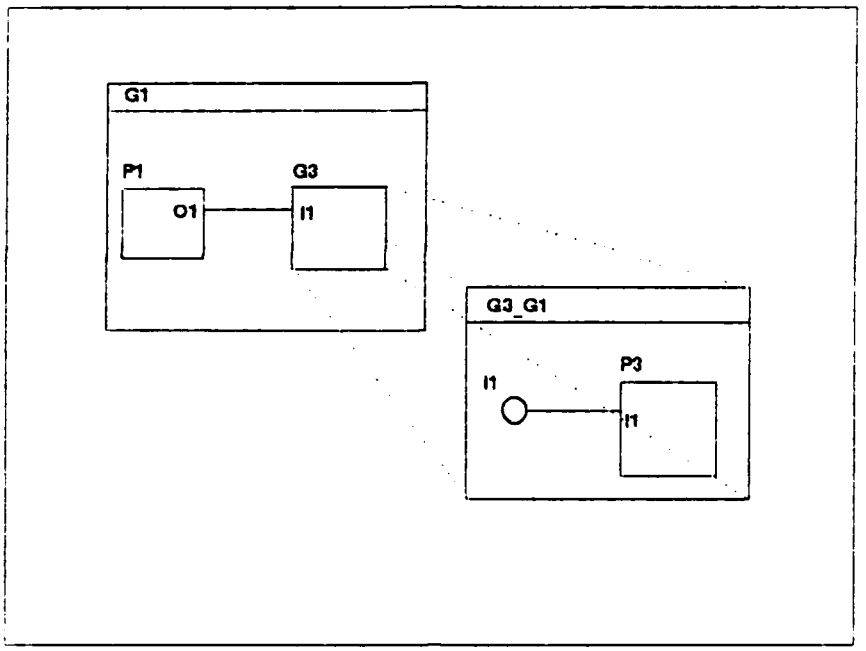


Figure 4.3 A User's View of a Dataflow Graph

the top of the window) has a primitive node P1 and a sub-graph node G3.

G3 is a graph. The window labeled G3_G1 (G3's full name) shows that G3 has an input connection, I1, which is connected to a primitive P3.

The data structure used to represent the simple design hierarchy is shown in Figure 4.4. There are three different types of elements in Figure 4.4. The rectangles are nodes of the functional network (primitive names begin with a "P" and graph names begin with a "G"). The ellipse shapes are connectors (input connector names begin with a "I" and output connectors names begin with an "O"). The "lozenge" shapes are arcs (arc names begin with an "A").

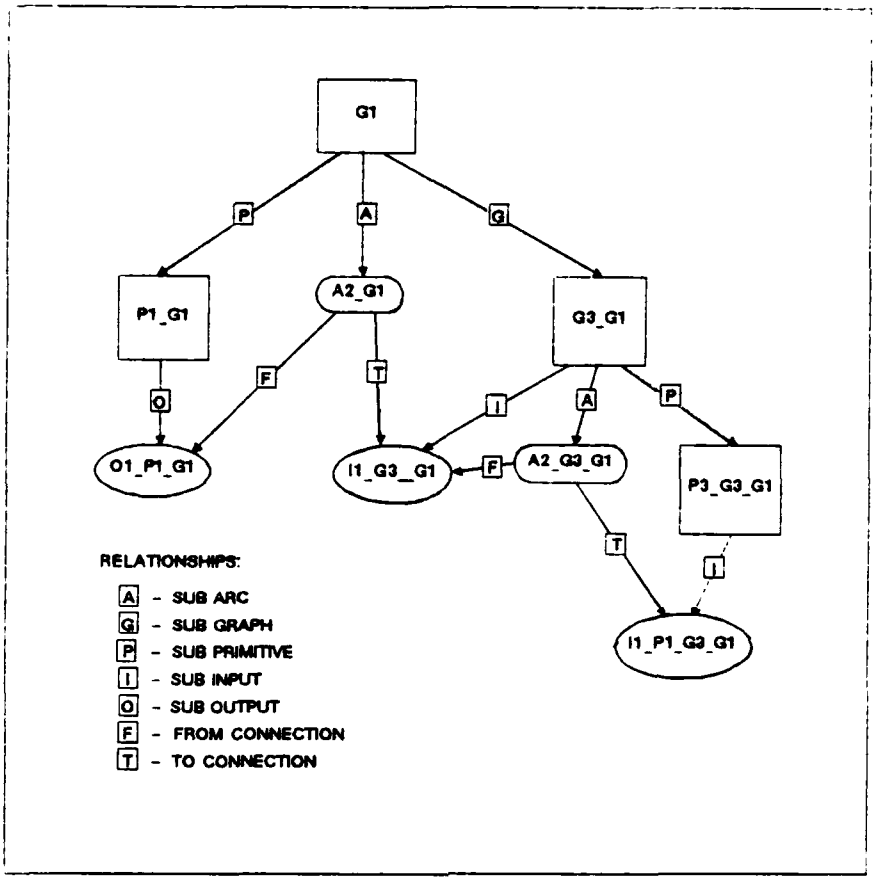


Figure 4.4 Data Structure for the Dataflow Graph

The three types of elements in Figure 4.4 are connected with directed lines which represent different relationships between the elements. Most of the relationships in Figure 4.4 are "sub-element" relationships. For example, the graph G3_G1 is the SUB-GRAPH of G1. The exceptions to the sub-element relationships occur in conjunction with the arcs. Arc elements use a FROM-CONNECTOR and a TO-CONNECTOR relationship to show were the arc begins and ends.

The in the implementation of GDA, a frame-based knowledge representation tool (KEE) was used to represent

dataflow graphs. The different types of elements (nodes, arcs, and connectors) are represented with units. (A unit is a frame in KEE). Units have slots which can contain pointers to other units. The relationships shown in Figure 4.4 were represented with slots. The P1_G1 primitive unit, for example, has a slot named SUB-OUTPUT that contains a pointer to O1_P1_G1.

The names used for elements Figure 4.4 are longer than the names used by the user (Figure 4.3). It was necessary to design a naming convention to insure that each unit in the KEE knowledge base has a unique name. The first part of each name consists of the name used by the user, followed by the path (following the sub-element relations) to the top of the tree. The primitive P3 (Figure 4.3) has a long name of P3_G3_G1 because P3 is a SUB-PRIMITIVE of G3, which is a SUB-GRAPH of G1.

The dataflow graph shown in Figure 4.4 is part of the USER-GRAPHS knowledge-base. The classes, which contain the definitions for the elements in the dataflow graph, are contained in the GDA-SYSTEM and FGN-GRAMMAR knowledge bases. The next section discusses the knowledge bases which comprise the Graph Design Assistant.

4.2 GDA knowledge bases

As was previously mentioned, the Graph Design Assistant was implemented by building KEE knowledge bases. The two major knowledge bases which define the Graph Design Assistant are the GDA-SYSTEM and the FGN-GRAMMAR knowledge bases (see Figure 3.2 and Figure 3.3). The other knowledge base, USER-GRAPHS, is not part of GDA's definition, but contains the dataflow graphs the user is currently editing.

This sections examines GDA-SYSTEM and FGN-GRAMMAR in more detail. Specifically, some of the individual units within each knowledge base are described. The purpose of this section is to show some of the techniques used in the implementation of GDA. It is not intended to exhaustively describe all of the units and all of their slots. A listing of the complete contents of the knowledge bases can be found in Appendix A.

The GDA-SYSTEM knowledge base contains the "generic" knowledge about dataflow graphs. The FGN-GRAMMAR knowledge base, on the other hand, contains knowledge about a specific dataflow language, the Functional Graph Network (FGN) language.

One of the important classes of objects defined in GDA-SYSTEM is a "graph element". The graph elements are objects which are used to construct the representation of a dataflow graph that resides in the USER-GRAPHS knowledge base.

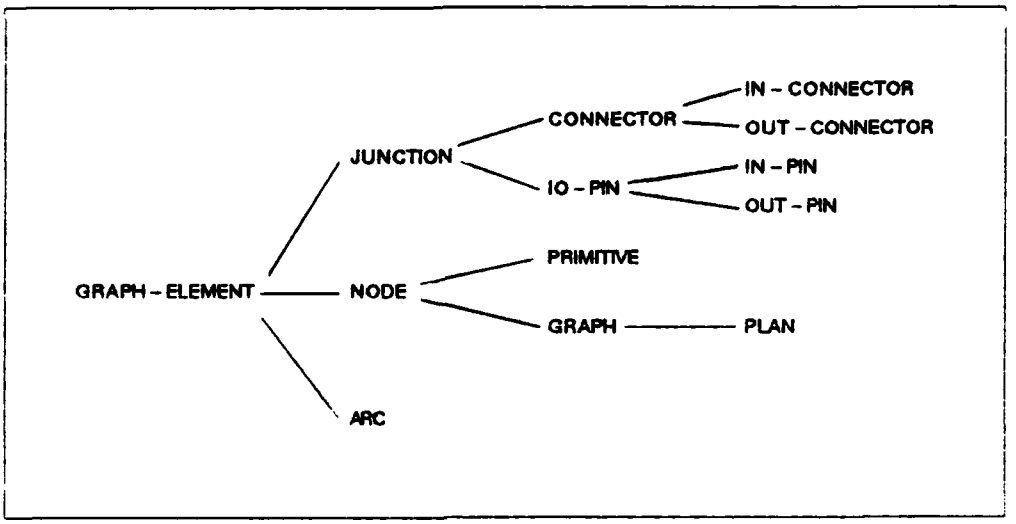


Figure 4.5 Graph Element Class Hierarchy

The graph elements are defined with a hierarchy of classes. Figure 4.5 shows the classes used to define the graph elements. Each class shown in the tree is a separate unit (or frame) within the GDA-SYSTEM knowledge base. The top unit of the hierarchy is GRAPH-ELEMENT. The lines in Figure 4.5 indicate sub-class relationships. Therefore, JUNCTION, NODE, and are sub-classes of GRAPH-ELEMENT.

The purpose of the hierarchy is the common properties can be "inherited" rather than defined separately within each unit (Intellicorp, 1985d:5.1-5.7). An inherited property can be overridden by defining that property at a lower level. In other words, the inherited property is a default, but may be replaced if necessary.

The properties of GRAPH-ELEMENT are the most general type and defined at that level. At lower levels in the

hierarchy (to the right in Figure 4.5) the properties are more specific to the particular type of element. For example, all of the elements of a graph have a FIRSTNAME; therefore, that property is defined in the GRAPH-ELEMENT unit and inherited by all the other sub-classes. The NODE class contains a SUB-INPUT property which is common to all of it's sub-classes (PRIMITIVE and GRAPH), but is not relevant the the ARC or JUNCTION classes.

The properties of a class are represented with "slots". A slot can contain data values, "methods" (a procedure or function), "active values" (a combination of a data value and a procedure) (Intellicorp, 1985d:11.1-11.7,12.1-12-8).

In order understand the complete definition of graph element objects, it is necessary to examine the units and slots of the classes, as well as the class hierarchy. What follows is a description of some of the units and their slots in the class hierarchy shown in Figure 4.5.

GRAPH-ELEMENT unit. The GRAPH-ELEMENT is the most "generic" class description of the elements which comprise a dataflow graph. All of the graph elements are described by sub-classes of the GRAPH-ELEMENT unit.

One of the data slots is FIRSTNAME. This is used with a method slot NAME-ELEMENT. An element is named by sending a NAME-ELEMENT message to that element. The complete name for the element is created by appending the FIRSTNAME to the name of the element's SUPER-ELEMENT.

All elements have a method slot named DRAW. If an element receives a DRAW message, that method knows how to draw a graphic representation of itself in a window. All elements inherit the DRAW slot, however, the value of the slot is overridden at the lower levels because the different types of elements have have different graphic representations. The immediate sub-classes to GRAPH-ELEMENT, JUNCTION, NODE, and ARC, all display themselves differently.

Some of the methods defined in GRAPH-ELEMENT have to do with the construction and modification of the dataflow graph representation. INSTANTIATE is a method slot in GRAPH-ELEMENT. When a class receives an INSTANTIATE message, it creates an instance of that class. That instance is the graph element which becomes part of the dataflow graph. DELETE is a method slot in GRAPH-ELEMENT. When an element receives a DELETE message it deletes all of its sub-elements (by sending them DELETE messages), removes pointers to itself from its super-element, and deletes itself from the USER-GRAPHS knowledge base.

JUNCTION unit. The JUNCTION class describes objects which serve as the connection points within dataflow graphs. The DELETE slot in JUNCTION illustrates a variation of the normal inheritance mechanism. Normally when a slot value is defined, its new value simply overrides inherited values. The DELETE method which was inherited from GRAPH-ELEMENT was applicable to JUNCTION objects, however it needed

modification to work correctly. KEE allows the definition of "before" and "after" methods (Intellicorp, 1985:11.7-11.8). In definition of JUNCTION's DELETE slot, a "before" method was added to delete IN-ARC and OUT-ARC pointers. This section of code is executed before the method inherited from GRAPH-ELEMENT. The "before" and "after" methods allows the modifications of procedures as they are inherited from sub-class to sub-class.

4.3 Graph-Window and GDA-Interface

The principle means of the interfacing GDA with the user is through a "graph-window" and its associated "GDA-interface" process (Figure 3.2). This section describes some of the implementation details for the graph-window and the GDA-Interface. The descriptions are not meant to be exhaustive (the source code is found in Appendix B).

The graph-window appears as rectangular area on the screen of the Symbolics 3600. When a graph-window is created by the user, a GDA-interface process is started. The user interacts with the graph-window by selecting menu items with the mouse. The GDA-interface process receives the menu selections and executes the user's commands.

The graph-window and GDA-interface were implemented outside of the KEE environment. This is possible because

KEE allows complete access to the underlying Symbolic's Zetalisp programming environment. The reason that the decision was made to implement the graph-window outside of KEE, is that the windows used by KEE did not support the properties that were needed for the interface.

Graph-window interface behavior. The graph-window interface was design to display a dataflow graphs. The user makes enters commands to modify graph elements by pointing to the element and clicking a mouse button. There are actually two active buttons for the graph-window interface. The left button always executes a default command for the object, while the right button brings up a "pop-up" menu of other commands applicable to the object.

Along the top of the graph-window, run a row of menu items that the user can select. These invoke commands which are applicable to objects that are not visible in the window or commands which apply to the window itself.

The Graph-window-flavor. The windows facility in the Symbolics Zetalisp environment is implemented using Zetalisp's "Flavors" system (Symbolics 1985d: 207-255). Flavors is an object oriented programming system where first the flavor, of an object is defined, then the object is instantiated. Objects are sent messages with cause the execution of "methods".

The graph-window was created by first defining a flavor of window called "graph-window-flavor". Several predefined

Zetalisp flavors as well as an existing KEE window flavor were "mixed" to produce the graph-window-flavor. The resulting graph-window-flavor had all of the properties (methods and variables) of the Zetalisp and KEE windows. In addition, the graph-window-flavor was customized further by defining new methods.

V. Results, Conclusions, and Recommendations

5.1 Results

This thesis effort resulted in the design and development of a prototype programming tool for the development of dataflow programs. The version of the Graph Design Assistant should serve as the basis for further development of GDA. In addition, GDA can serve as a model for similar "visual programming" tools applied to different programming problems.

Two specific results were the design of a representation for dataflow graphs and the development of an user interface that can create and edit dataflow graphs.

Not all of the original goals of the effort were met. The intention was to build a prototype that contained all of the components of the complete GDA system (even if some of the parts were not completely developed). The actual implementation, however, did not contain all of the features envisioned in the original design. There is only rudimentary error checking of the dataflow graphs, and the implementation of the "plan" creation and editing capability was not completed.

5.2 Conclusions

The "programmer's assistant" and the "visual programming" approach taken by GDA appears to be a feasible way to create dataflow program. Although, the prototype did not implement all of the features necessary for a robust programming tool, there did not appear to be any insurmountable difficulties in building a complete system.

The strategy used to develop GDA was an important factor in making it possible to implement the system in a short time. GDA was developed with a commercially available "knowledge engineering tool" (KEE). During the short development time (approximately 3 man-months) a working prototype was developed. In the judgment of the author, had the development effort been attempted with a less powerful programming environment, much less progress would have been possible.

5.3 Recommendations

Continued development of the Graph Design Assistant. There are several problems that were not adequately addressed in the present implementation of GDA. The representation of "plans" needs to be investigated. This is the means of storing domain knowledge in GDA. Plans are selected from the a library by the user and are instantiated

to form parts of graphs. The representation of plans should allow for the encoding of knowledge that affects how the plan expands. For example, a plan that controls a joint on a robot should be able to build a network that incorporates constraints that apply to the joint. The joint may be limited in the speed and the angles it can rotate. The functional network that controls the joint can have those limits "built in" by the plan when it is instantiated.

Evaluation of the Graph Design Assistant. At some point, an effort to evaluate the effectiveness of GDA should be made. This could be done by comparing the experience of users who learn to program the PS300 directly by writing text FGN programs with the experience of users who learn to program the PS300 using the GDA system.

Other dataflow "grammars". The present implementation of the Graph Design Assistant contains representations for one particular type of dataflow graph. It was design specifically for the grammar of a special-purpose Functional Graph Network (FGN) language used to program the Evans and Sutherland PS300. GDA was designed, however, so that the FGN grammar could be replaced descriptions of other dataflow languages.

One possibility for another dataflow grammar which could be used in GDA is one that describes the robot simulation facility. In the chapter 2, the requirements for the robot simulation facility were specified with a dataflow

diagram (Figure 2.1). It should be possible to define the process (nodes of the dataflow diagram) and the data which passes between the process as elements of the grammar. The objective would be to allow the user to dynamically configure the robot simulations using by editing graphs with GDA.

An even more general dataflow grammar could be design to configure "real-time" systems. Dataflow models have been proposed as a method of specifying real-time systems (Allworth, 1981:13-24). It would be interesting to see if the capabilities of GDA could be enhanced to allow a user to build real-time systems by editing dataflow graphs.

Appendix A: Graph Design Assistant Knowledge Bases

The section contains listings of the KEE knowledge bases which define the Graph Design Assistant. Each knowledge base listing begins with the name of the knowledge base followed a list of the knowledge base's contents. The knowledge base contains units. Each unit and it's slots are listed.

Knowledge Base: GDA-SYSTEM

Contents:

ADD-ONLY-INPUTS
ADD-ONLY-OUTPUTS
ARC
CONNECTOR
CREATE-GRAPH-WINDOW
CREATE-TREE-WINDOW
DATA-TOKEN-TYPE
GDA-ACTIVEVALUES
GDA-COMMANDS
GDA-SYSTEM-UPDATE-COMMANDS
GET-LIBRARY-MEMBERS-MENU
GRAMMAR
GRAPH
GRAPH-ELEMENT
IMAGE.PANEL@@443
IMAGES
IN-CONNECTOR
IN-PIN
IO-PIN
JUNCTION
LIBRARY
METHOD.ACTUATOR@@474
METHOD.ACTUATOR@@499
METHOD.ACTUATOR@@577
NODE
OUT-CONNECTOR
OUT-PIN
PLAN
PLAN-LIBRARY
PRIMITIVE
PRIMITIVE-LIBRARY
UPDATE-SUB-ELEMENT-POINTER

```
Unit: ADD-ONLY-INPUTS
  Members: NIL
  Subclasses: NIL

  Own slot: AVPUT
    Valueclass: NIL
    Values: (LAMBDA (SELF SLOT NEWVALUE OLDVALUE UNIT SLOTTYPE)
          (IF (GET.VALUE UNIT 'INPUT-P) NEWVALUE OLDVALUE))


Unit: ADD-ONLY-OUTPUTS
  Members: NIL
  Subclasses: NIL


Unit: ARC
  Members: NIL
  Subclasses: NIL

  Member slot: DEFAULT-NAME-PREFIX
    Valueclass: NIL
    Values: (A)

  Member slot: DELETE
    Valueclass: NIL
    Values: ((BEFORE (LET* ((TO-JUNCTION (GET.VALUE SELF 'TO-JUNCTION))
                            (FROM-JUNCTION (GET.VALUE SELF 'FROM-JUNCTION)))
                  (REMOVE.VALUE FROM-JUNCTION 'OUT-ARC SELF)
                  (REMOVE.VALUE TO-JUNCTION 'IN-ARC SELF)))
       )

  Member slot: DRAW
    Valueclass: NIL
    Values: (LAMBDA (SELF WINDOW)
          (SEND WINDOW :DRAW-ARC SELF))
```

```
  Member slot: FROM-JUNCTION
    Valueclass: (JUNCTION)
    Values: NIL

  Member slot: PARENT-SUB-POINTER-SLOT
    Valueclass: NIL
    Values: (SUB-ARC)

  Member slot: PATH
    Valueclass: NIL
    Values: NIL

  Member slot: TO-JUNCTION
    Valueclass: (JUNCTION)
    Values: NIL


Unit: CONNECTOR
  Members: NIL
  Subclasses: (#Unit (OUT-CONNECTOR GDA-SYSTEM) #Unit (IN-CONNECTOR GDA-SYSTEM))

  Member slot: CONNECT-POINT-OFFSET
    Valueclass: NIL
    Values: NIL

  Member slot: DRAW
    Valueclass: NIL
    Values: (LAMBDA (SELF WINDOW)
          (SEND WINDOW :DRAW-CONNECTOR SELF))

  Member slot: IN-ARC
    Valueclass: (ARC)
    Values: NIL

  Member slot: INPUT-P
    Valueclass: ((ONE-OF T NIL))
    Values: NIL
```

```
  Member slot: LABEL
    Valueclass: (STRING)
    Values: NIL

  Member slot: LABEL-OFFSET
    Valueclass: NIL
    Values: ((0 0))

  Member slot: OUT-ARC
    Valueclass: (ARC)
    Values: NIL

  Member slot: SHAPE
    Valueclass: NIL
    Values: NIL


Unit: CREATE-GRAPH-WINDOW
  Members: NIL
  Subclasses: NIL

  Own slot: DO
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF)
          (START-GRAPH-WINDOW-PROCESS))

  Own slot: \UNIT.IMAGE
    Valueclass: NIL
    Values: NIL


Unit: CREATE-FREE-WINDOW
  Members: NIL
  Subclasses: NIL

  Own slot: DO
    Valueclass: NIL
    Values: (LAMBDA (SELF)
          (DECLARE (SPECIAL KB))
          (SETQ KB NIL)
          (TV:CHOOSE-VARIABLE-VALUES '((KB "Knowledge Base"))
                                     :LABEL
```

```
                                        "Enter knowledge base to display:")
            (SLOT.GRAPH.KB KB 'SUB-ELEMENT NIL (CREATE.KEE.WINDOW) '(HORIZONTAL.))


Unit: DATA-TOKEN-TYPE
  Members: NIL
  Subclasses: (#Unit (FGN-DATA-TOKEN-TYPE FGN-GRAMMAR-KB))


Unit: GDA-ACTIVEVALUES
  Members: (#Unit (ADD-ONLY-OUTPUTS GDA-SYSTEM)
     #Unit (ADD-ONLY-INPUTS GDA-SYSTEM)
     #Unit (UPDATE-SUB-ELEMENT-POINTER GDA-SYSTEM)
     #Unit (GET-LIBRARY-MEMBERS-MENU GDA-SYSTEM))
  Subclasses: NIL


Unit: GDA-COMMANDS
  Members: (#Unit (CREATE-TREE-WINDOW GDA-SYSTEM) #Unit (CREATE-GRAPH-WINDOW GDA-SYSTEM))
  Subclasses: NIL

  Member slot: COMMAND-MENU
    Valueclass: (MENUITEM)
    Values: (("CREATE-TREE-WINDOW"
        #Unit (CREATE-TREE-WINDOW GDA-SYSTEM)
        "Creates a window that shows a hierchical display of graphs in a kb")
       ("CREATE-GRAPH-WINDOW" #Unit (CREATE-GRAPH-WINDOW GDA-SYSTEM) NIL))

  Member slot: DO
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF))

  Member slot: SELECT-COMMAND
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF)
          (LET ((COMMAND-UNIT (MENU (GET.SLOT.MENU SELF 'COMMAND-MENU))))
            (IF COMMAND-UNIT (UNITMSG COMMAND-UNIT 'DO))))

  Member slot: UPDATE-COMMAND-MENU
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF)
          (LET* ((COMMAND-UNITS (UNIT.ALLCHILDREN SELF 'MEMBER)))
            (REMOVE.ALL.LOCAL.VALUES SELF 'COMMAND-MENU)
            (DOLIST (CU COMMAND-UNITS)
```

```
                    (LET ((TEXT (FORMAT NIL "~a" (UNIT.NAME CU)))
                          (VALUE CU)
                          (PROMPT (UNIT.COMMENT CU)))
                      (ADD.VALUE SELF
                                 'COMMAND-MENU
                                 '(,TEXT ,VALUE ,PROMPT))))))

  Own slot: \IMAGE.PANEL
    Valueclass: NIL
    Values: (#Unit (IMAGE.PANEL@@443 GDA-SYSTEM))


Unit: GDA-SYSTEM-UPDATE-COMMANDS
  Members: NIL
  Subclasses: NIL

  Member slot: CREATE.INSTANCE
    Valueclass: NIL
    Values: NIL

  Member slot: CREATE.PROGRAMMATICALLY
    Valueclass: NIL
    Values: CREATE.COMPOSITE.CLASS.PROGRAMMATICALLY

  Own slot: GERM
    Valueclass: NIL
    Values: (NIL #Unit (IMAGE.PANEL ACTIVEIMAGES)
            NIL
            (368 602 348 147)
            (109 519 347 48)
            EXPAND
            NIL
            ((NIL #Unit (METHOD.ACTUATOR ACTIVEIMAGES)
                  NIL
                  (384 669 120 51)
                  (384 669 209 48)
                  EXPAND
                  NIL
                  NIL
                  NIL)
             )
            NIL)
```

```
  Own slot: \IMAGE.PANEL
    Valueclass: NIL
    Values: NIL


Unit: GET-LIBRARY-MEMBERS-MENU
  Members: NIL
  Subclasses: NIL

  Own slot: AVGET
    Valueclass: NIL
    Values: (LAMBDA
       (SELF SLOT VALUE UNIT SLOTTYPE)
       (LET
        ((SUBITEMS (UNIT.CHILDREN UNIT 'SUBCLASS))
         (IMMEDIATE-MEMBERS (UNIT.CHILDREN UNIT 'MEMBER)))
        (APPEND
         (MAPCAR (FUNCTION (LAMBDA (SUBITEM)
                           (LET ((MENU-TEXT (FORMAT NIL
                                                    "~a"
                                                    (UNIT.NAME SUBITEM)))
                                 (VALUE-RETURNED SUBITEM)
                                 (PROMPT-MESSAGE (GET.VALUE SUBITEM
                                                            'PROMPT-MESSAGE))
                                 (SUB-MENU (GET.VALUE SUBITEM 'MEMBER-MENU)))
                             '(,MENU-TEXT ,VALUE-RETURNED ,PROMPT-MESSAGE
                               (SUBITEMS ,SUB-MENU)))))
                 SUBITEMS)
         (MAPCAR
          (FUNCTION (LAMBDA (MEMBER)
                      (LET ((MENUTEXT (FORMAT NIL "~a" (UNIT.NAME MEMBER)))
                            (VALUE-RETURNED MEMBER)
                            (PROMPT-MESSAGE (GET.VALUE MEMBER
                                                       'PROMPT-MESSAGE)))
                        '(,MENUTEXT ,VALUE-RETURNED ,PROMPT-MESSAGE))))
          IMMEDIATE-MEMBERS))))


Unit: GRAMMAR
  Members: (#Unit (FGN-GRAMMAR FGN-GRAMMAR-KB))
  Subclasses: NIL
```

```
  Member slot: PLAN-LIBRARY
    Valueclass: ((SUBCLASS.OF PLAN-LIBRARY))
    Values: NIL

  Member slot: PRIMITIVE-LIBRARY
    Valueclass: ((SUBCLASS.OF PRIMITIVE-LIBRARY))
    Values: NIL


Unit: GRAPH
  Members: NIL
  Subclasses: (#Unit (PLAN GDA-SYSTEM))

  Member slot: ADD-SUB-ELEMENT
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF CLASS)
          (LET* ((NEWUNIT (UNITMSG CLASS 'INSTANTIATE))
                 (PREFIX (GET.VALUE CLASS 'DEFAULT-NAME-PREFIX))
                 (NEWNAME (UNITMSG SELF 'GENERATE-SUB-NAME PREFIX))
                 (PARENT-SUB-POINTER-SLOT (GET.VALUE NEWUNIT
                                                     'PARENT-SUB-POINTER-SLOT)))
            (PUT.VALUE NEWUNIT 'SUPER-ELEMENT SELF)
            (UNITMSG NEWUNIT 'NAME-ELEMENT NEWNAME)
            (ADD.VALUE SELF PARENT-SUB-POINTER-SLOT NEWUNIT)
            NEWUNIT
            ))

  Member slot: CHECK
    Valueclass: NIL
    Values: (LAMBDA (SELF AGENDA)
          (LET ((SUB-ELEMENTS (GET.VALUE SELF 'SUB-ELEMENTS)))
            (DOLIST (SE SUB-ELEMENTS) (UNITMSG SE 'CHECK AGENDA))))

  Member slot: DEFAULT-NAME-PREFIX
    Valueclass: NIL
    Values: (SG)

  Member slot: DRAW-SUB-ELEMENTS
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF WINDOW)
          (LET* ((PRIMITIVES (GET.VALUES SELF 'SUB-PRIMITIVE))
                 (GRAPHS (GET.VALUES SELF 'SUB-GRAPH))
                 (INPUTS (GET.VALUES SELF 'SUB-INPUT))
                 (OUTPUTS (GET.VALUES SELF 'SUB-OUTPUT))
```

```
            (ARCS (GET.VALUES SELF 'SUB-ARC)))
        (DOLIST (SUB (APPEND PRIMITIVES GRAPHS INPUTS OUTPUTS ARCS))
            (UNITMSG SUB 'DRAW WINDOW))))

Member slot: GENERATE-SUB-NAME
  Valueclass: (METHOD)
  Values: (LAMBDA (SELF PREFIX)
      (LET ((COUNT (1+ (GET.VALUE SELF 'SUB-ELEMENT-COUNT))))
        (PUT.VALUE SELF 'SUB-ELEMENT-COUNT COUNT)
        (MAKE-SYMBOL (FORMAT NIL "~a~d" PREFIX COUNT))))

Member slot: GRAMMAR
  Valueclass: (GRAMMAR)
  Values: (FGN-GRAMMAR)

Member slot: PARENT-SUB-POINTER-SLOT
  Valueclass: NIL
  Values: (SUB-GRAPH)

Member slot: REMOVE-SUB-ELEMENT
  Valueclass: (METHOD)
  Values: (LAMBDA (SELF SUBELEMENT))

Member slot: SELECTABLE-ITEM-TYPE
  Valueclass: NIL
  Values: (:GRAPH-ITEM)

Member slot: SUB-ARC
  Valueclass: (ARC)
  Values: NIL

Member slot: SUB-ELEMENT-COUNT
  Valueclass: (INTEGER)
  Values: (0)

Member slot: SUB-GRAPH
  Valueclass: (GRAPH)
  Values: NIL
```

```
  Member slot: SUB-POINTERS
    Valueclass: NIL
    Values: (SUB-ARC SUB-GRAPH SUB-PRIMITIVE)

  Member slot: SUB-PRIMITIVE
    Valueclass: (PRIMITIVE)
    Values: NIL


Unit: GRAPH-ELEMENT
  Members: NIL
  Subclasses: (#Unit (ARC GDA-SYSTEM) #Unit (NODE GDA-SYSTEM) #Unit (JUNCTION GDA-SYSTEM))

  Member slot: CHECK
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF AGENDA))

  Member slot: DEFAULT-NAME-PREFIX
    Valueclass: NIL
    Values: (GE)

  Member slot: DELETE
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF)
          (DOLIST (SUB (GET.VALUES SELF 'SUB-ELEMENT)) (UNITMSG SUB 'DELETE))
          (LET ((PARENT (GET.VALUE SELF 'SUPER-ELEMENT)))
            (IF PARENT
                (LET ((SUB-POINTER-SLOT (GET.VALUE SELF 'PARENT-SUB-POINTER-SLOT)))
                  (IF SUB-POINTER-SLOT (REMOVE.VALUE PARENT SUB-POINTER-SLOT SELF)))))
          (UNITDELETE SELF))

  Member slot: DRAW
    Valueclass: (METHOD)
    Values: NIL

  Member slot: FIRSTNAME
    Valueclass: NIL
    Values: NIL
```

```
Member slot: INSTANTIATE
  Valueclass: (METHOD)
  Values: (LAMBDA (SELF)
        (CREATE.UNIT (GENSYM 'E) NIL NIL SELF))

Member slot: NAME-ELEMENT
  Valueclass: (METHOD)
  Values: (LAMBDA (SELF &OPTIONAL NEWFIRSTNAME)
        (LET* ((FIRSTNAME (IF NEWFIRSTNAME
                              NEWFIRSTNAME
                              (GET.VALUE SELF 'FIRSTNAME)))
               (OLDNAME (UNIT.NAME SELF))
               (PARENT (GET.VALUE SELF 'SUPER-ELEMENT))
               (NEWNAME (IF PARENT
                            (MAKE-SYMBOL (FORMAT NIL
                                                 "~a_~a"
                                                 FIRSTNAME
                                                 (UNIT.NAME PARENT)))
                            FIRSTNAME)))
          (COND ((NEQ NEWNAME OLDNAME)
                 (UNITRENAME SELF NEWNAME)
                 (PUT.VALUE SELF 'FIRSTNAME FIRSTNAME)
                 (DOLIST (CHILD (GET.VALUES SELF 'SUB-ELEMENT))
                         (UNITMSG CHILD 'NAME-ELEMENT))))
          NEWNAME
          ))

Member slot: PARENT-SUB-POINTER-SLOT
  Valueclass: ((ONE.OF SUB-INPUT SUB-OUTPUT SUB-GRAPH SUB-PRIMITIVE SUB-ARC))
  Values: NIL

Member slot: SELECTABLE-ITEM-TYPE
  Valueclass: ((ONE.OF :PRIMITIVE-ITEM :GRAPH-ITEM :INPUT-ITEM :OUTPUT-ITEM))
  Values: NIL

Member slot: SUB-POINTERS
  Valueclass: NIL
  Values: NIL

Member slot: SUPER-ELEMENT
  Valueclass: (NODE)
  Values: NIL
```

```
Unit: IMAGE.PANEL@@443
  Members: NIL
  Subclasses: NIL

  Own slot: BORDER
    Valueclass: NIL
    Values: 12

  Own slot: HEIGHT
    Valueclass: NIL
    Values: 111

  Own slot: IMAGE.WAS.PAINTED
    Valueclass: NIL
    Values: (NIL)

  Own slot: IMAGES
    Valueclass: NIL
    Values: (#Unit (METHOD.ACTUATOR@@577 GDA-SYSTEM)
       #Unit (METHOD.ACTUATOR@@474 GDA-SYSTEM))

  Own slot: OBJECT.DISPLAYED
    Valueclass: NIL
    Values: #KB (GDA-SYSTEM)

  Own slot: REGION
    Valueclass: NIL
    Values: (325 623 338 111)

  Own slot: TITLE
    Valueclass: NIL
    Values: "GDA User Commands"

  Own slot: TITLEFONT
    Valueclass: NIL
    Values: FONTS:HL1@@
```

```
  Own slot: TOPUNIT
    Valueclass: NIL
    Values: #Unit (GDA-COMMANDS GDA-SYSTEM)

  Own slot: WIDTH
    Valueclass: NIL
    Values: 338

  Own slot: WINDOW
    Valueclass: NIL
    Values: #<IWIN Iwin 4 16313352 deexposed>


Unit: IMAGES
  Members: (#Unit (IMAGE.PANEL00443 GDA-SYSTEM) #Unit (METHOD.ACTUATOR00577 GDA-SYSTEM)
                                                #Unit (METHOD.ACTUATOR00499 GDA-SYSTEM)
                                                #Unit (METHOD.ACTUATOR00474 GDA-SYSTEM))

  Subclasses: NIL

  Own slot: DELETE
    Valueclass: METHOD
    Values: DELETE.ALL.IMAGES

  Own slot: DELETE.ALL.IMAGES
    Valueclass: METHOD
    Values: DELETE.ALL.IMAGES

  Own slot: DONT.RECREATE.IMAGES.AFTER.KBLOAD
    Valueclass: (ONE.OF T NIL ASKUSER)
    Values: ASKUSER

  Own slot: RECREATE.ALL.IMAGES
    Valueclass: METHOD
    Values: RECREATE.ALL.IMAGES

  Own slot: SAVE.ALL.IMAGES
    Valueclass: METHOD
    Values: SAVE.ALL.IMAGES
```

```
  Own slot: USER.DELETE.ALL.IMAGES
    Valueclass: METHOD
    Values: USER.DELETE.ALL.IMAGES

  Own slot: USER.RECREATE.ALL.IMAGES
    Valueclass: METHOD
    Values: USER.RECREATE.ALL.IMAGES


Unit: IN-CONNECTOR
  Members: NIL
  Subclasses: NIL

  Member slot: CONNECT-POINT-OFFSET
    Valueclass: NIL
    Values: ((28 6))

  Member slot: DEFAULT-NAME-PREFIX
    Valueclass: NIL
    Values: (I)

  Member slot: INPUT-P
    Valueclass: NIL
    Values: (T)

  Member slot: LABEL
    Valueclass: NIL
    Values: ("in")

  Member slot: LABEL-OFFSET
    Valueclass: NIL
    Values: ((4 1))

  Member slot: PARENT-SUB-POINTER-SLOT
    Valueclass: NIL
    Values: (SUB-INPUT)
```

```
  Member slot: SHAPE
    Valueclass: NIL
    Values: ((0 0) (25 0) (28 6) (25 12) (0 12) (0 0))


Unit: IN-PIN
  Members: NIL
  Subclasses: (#Unit (FGN-IN-PIN FGN-GRAMMAR-KB))

  Member slot: CONSTANT-INPUT
    Valueclass: ((ONE.OF T NIL))
    Values: NIL

  Member slot: DEFAULT-NAME-PREFIX
    Valueclass: NIL
    Values: (I)

  Member slot: IN-ARC
    Valueclass: (ARC)
    Values: NIL

  Member slot: PARENT-SUB-POINTER-SLOT
    Valueclass: NIL
    Values: (SUB-INPUT)

  Member slot: SELECTABLE-ITEM-TYPE
    Valueclass: NIL
    Values: (:INPUT-ITEM)


Unit: IO-PIN
  Members: NIL
  Subclasses: (#Unit (IN-PIN GDA-SYSTEM) #Unit (OUT-PIN GDA-SYSTEM))

  Member slot: LEGAL-DATA-TOKENS
    Valueclass: (DATA-TOKEN-TYPE)
    Values: NIL
```

```
Unit: JUNCTION
  Members: NIL
  Subclasses: (#Unit (IO-PIN GDA-SYSTEM) #Unit (CONNECTOR GDA-SYSTEM))

  Member slot: CALC-DISPLAY-HEIGHT
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF WINDOW))

  Member slot: CALC-DISPLAY-WIDTH
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF WINDOW))

  Member slot: CONNECT-POINT
    Valueclass: NIL
    Values: NIL

  Member slot: DELETE
    Valueclass: NIL
    Values: ((BEFORE (LET ((IN-ARCS (GET.VALUES SELF 'IN-ARC))
                           (OUT-ARCS (GET.VALUES SELF 'OUT-ARC)))
                       (DOLIST (IN IN-ARCS) (UNITMSG IN 'DELETE))
                       (DOLIST (OUT OUT-ARCS) (UNITMSG OUT 'DELETE))))
             )

  Member slot: INPUT-POSITION
    Valueclass: NIL
    Values: NIL

  Member slot: OUTPUT-POSITION
    Valueclass: NIL
    Values: NIL

  Member slot: POSITION
    Valueclass: NIL
    Values: ((0 20))
```

```
  Member slot: SELECT-ARC-FROM-SLOT
    Valueclass: (METHOD)
    Values: (LAMBDA
       (SELF SLOT &OPTIONAL LABEL)
       (LET
        ((ARCS (GET.VALUES SELF SLOT)))
        (IF
         (< 1 (LENGTH ARCS))
         (LET
          ((ALIST
            (MAPCAR
             (FUNCTION (LAMBDA (ARC)
                         (CONS (LET* ((FROM (GET.VALUE ARC 'FROM-JUNCTION))
                                      (TO (GET.VALUE ARC 'TO-JUNCTION))
                                      (TO-NODE (GET.VALUE TO 'SUPER-ELEMENT)))
                                 (FORMAT NIL
                                         "from ~a to ~a,~a"
                                         (GET.VALUE FROM 'FIRSTNAME)
                                         (GET.VALUE TO 'FIRSTNAME)
                                         (GET.VALUE TO-NODE 'FIRSTNAME)))
                               ARC)))
             ARCS)))
          (SELECT-FROM-ALIST ALIST LABEL))
         (CAR ARCS))))


Unit: LIBRARY
  Members: NIL
  Subclasses: (#Unit (PRIMITIVE-LIBRARY GDA-SYSTEM) #Unit (PLAN-LIBRARY GDA-SYSTEM))

  Member slot: GET-MENU-ITEM
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF)
          (LET ((TEXT (FORMAT NIL "~a" (UNIT.NAME SELF)))
                (VALUE SELF)
                (PROMPT (FORMAT NIL "Instantiate ~a" (UNIT.NAME SELF))))
            '(.TEXT .VALUE .PROMPT)))

  Member slot: GET-MENU-SUBITEMS
    Valueclass: (METHOD)
    Values: (LAMBDA
       (SELF)
       (LET*
        ((SUBCLASSES (REVERSE (UNIT.CHILDREN SELF 'SUBCLASS)))
         (MEMBERS (REVERSE (UNIT.CHILDREN SELF 'MEMBER)))
         (SUBCLASS-ITEMS (IF SUBCLASSES
                             (MAPCAR (FUNCTION (LAMBDA (SC)
```

```
                                                    (UNITMSG SC
                                                             'GET-MENU-SUBITEMS)))
                                        SUBCLASSES)))
            (MEMBER-ITEMS (IF MEMBERS
                              (MAPCAR (FUNCTION (LAMBDA (SC)
                                                        (UNITMSG SC 'GET-MENU-ITEM)))
                                      MEMBERS)))
            (TEXT-VALUE-PROMPT '(,(UNIT.NAME SELF) ,SELF "See sub menu")))
           (COND ((AND SUBCLASS-ITEMS MEMBER-ITEMS)
                  (APPEND TEXT-VALUE-PROMPT
                          (LIST (CONS 'SUBITEMS
                                      (APPEND SUBCLASS-ITEMS MEMBER-ITEMS)))))
                 (SUBCLASS-ITEMS
                  (APPEND TEXT-VALUE-PROMPT
                          '((SUBITEMS . ,SUBCLASS-ITEMS))))
                 (MEMBER-ITEMS
                  (APPEND TEXT-VALUE-PROMPT
                          '((SUBITEMS . ,MEMBER-ITEMS)))))))

Member slot: MEMBER-MENU
  Valueclass: (MENUITEM)
  Values: NIL

Member slot: MEMBER-MENU-TEST
  Valueclass: NIL
  Values: NIL

Member slot: SELECT-FROM-MENU
  Valueclass: (METHOD)
  Values: (LAMBDA (SELF)
            (MENU (GET.SLOT.MENU SELF 'MEMBER-MENU)))

Member slot: UPDATE-MEMBER-MENU
  Valueclass: (METHOD)
  Values: (LAMBDA (SELF)
            (LET ((SUBCLASSES (UNIT.CHILDREN SELF 'SUBCLASS))
                  (MEMBERS (UNIT.CHILDREN SELF 'MEMBER)))
              (REMOVE.ALL.LOCAL.VALUES SELF 'MEMBER-MENU)
              (DOLIST (SC SUBCLASSES)
                      (ADD.VALUE SELF 'MEMBER-MENU (UNITMSG SC 'GET-MENU-SUBITEMS)))
              (DOLIST (M MEMBERS)
                      (ADD.VALUE SELF 'MEMBER-MENU (UNITMSG M 'GET-MENU-ITEM)))))
```

```
Unit: METHOD.ACTUATOR00474
  Members: NIL
  Subclasses: NIL

  Own slot: BORDER
    Valueclass: NIL
    Values: 4

  Own slot: FONT
    Valueclass: NIL
    Values: FONTS:HL6

  Own slot: HEIGHT
    Valueclass: NIL
    Values: 36

  Own slot: IMAGE.WAS.PAINTED
    Valueclass: NIL
    Values: T

  Own slot: OBJECT.DISPLAYED
    Valueclass: NIL
    Values: #Slot (DO CREATE-GRAPH-WINDOW GDA-SYSTEM OWN)

  Own slot: REGION
    Valueclass: NIL
    Values: NIL

  Own slot: SUPER.IMAGE
    Valueclass: NIL
    Values: #Unit (IMAGE.PANEL00443 GDA-SYSTEM)

  Own slot: TITLE
    Valueclass: NIL
    Values: "Create a new Graph Window"
```

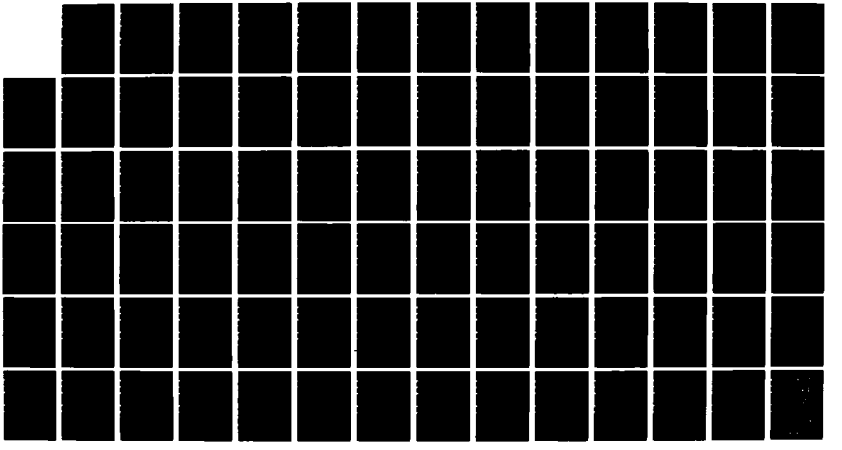
```
  Own slot: TITLEFONT
    Valueclass: NIL
    Values: FONTS:HL10B

  Own slot: TOPUNIT
    Valueclass: NIL
    Values: (NIL)

  Own slot: VALUE.WAS.SAVED
    Valueclass: NIL
    Values: T

  Own slot: WIDTH
    Valueclass: NIL
    Values: 182

  Own slot: WINDOW
    Valueclass: NIL
    Values: #<IWIN Iwin 8 16314026 deexposed>


Unit: METHOD.ACTUATOR00499
  Members: NIL
  Subclasses: NIL

  Own slot: BORDER
    Valueclass: NIL
    Values: 4

  Own slot: FONT
    Valueclass: NIL
    Values: FONTS:HL6

  Own slot: HEIGHT
    Valueclass: NIL
    Values: 51
```

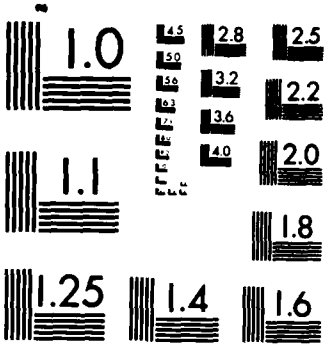
```
Own slot: IMAGE.WAS.PAINTED
  Valueclass: NIL
  Values: T

Own slot: OBJECT.DISPLAYED
  Valueclass: NIL
  Values: #Slot (SELECT-COMMAND GDA-COMMANDS GDA-SYSTEM MEMBER)

Own slot: REGION
  Valueclass: NIL
  Values: NIL

Own slot: TITLE
  Valueclass: NIL
  Values: "GDA-COMMANDS's SELECT-COMMAND"

Own slot: TITLEFONT
  Valueclass: NIL
  Values: FONTS:HL10B

Own slot: TOPUNIT
  Valueclass: NIL
  Values: (NIL)

Own slot: VALUE.WAS.SAVED
  Valueclass: NIL
  Values: T

Own slot: WIDTH
  Valueclass: NIL
  Values: 109

Own slot: WINDOW
  Valueclass: NIL
  Values: #<IWIN Iwin 2 16313124 deexposed>


Unit: METHOD.ACTUATOR@0577
  Members: NIL
  Subclasses: NIL
```

```
Own slot: BORDER
  Valueclass: NIL
  Values: 4

Own slot: FONT
  Valueclass: NIL
  Values: FONTS:HL10

Own slot: HEIGHT
  Valueclass: NIL
  Values: 41

Own slot: IMAGE.WAS.PAINTED
  Valueclass: NIL
  Values: T

Own slot: OBJECT.DISPLAYED
  Valueclass: NIL
  Values: #Slot (DO CREATE-TREE-WINDOW GDA-SYSTEM OWN)

Own slot: REGION
  Valueclass: NIL
  Values: NIL

Own slot: SUPER.IMAGE
  Valueclass: NIL
  Values: #Unit (IMAGE.PANEL00443 GDA-SYSTEM)

Own slot: TITLE
  Valueclass: NIL
  Values: "Create a window of showing sub-element relations"

Own slot: TITLEFONT
  Valueclass: NIL
  Values: FONTS:HL10B

Own slot: TOPUNIT
  Valueclass: NIL
  Values: (NIL)
```

```
  Own slot: VALUE.WAS.SAVED
    Valueclass: NIL
    Values: T

  Own slot: WIDTH
    Valueclass: NIL
    Values: 322

  Own slot: WINDOW
    Valueclass: NIL
    Values: #<IWIN Iwin 6 1631600 deexposed>


Unit: NODE
  Members: NIL
  Subclasses: (#Unit (GRAPH GDA-SYSTEM) #Unit (PRIMITIVE GDA-SYSTEM))

  Member slot: ADD-INITIAL-SUB-ELEMENT
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF FIRSTNAME CLASS SLOT-POINTER)
          (ADD.VALUE SELF
                     'INITIAL-SUB-ELEMENT
                     `(,FIRSTNAME ,(UNITREFERENCE CLASS) ,SLOT-POINTER)))

  Member slot: CALC-DISPLAY-HEIGHT
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF WINDOW)
          50)

  Member slot: CALC-DISPLAY-WIDTH
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF WINDOW)
          30)

  Member slot: DRAW
    Valueclass: (METHOD)
    Values: (LAMBDA (SELF WINDOW)
          (SEND WINDOW :DRAW-NODE SELF))
```

AD-A164 122 A PROGRAMMER'S ASSISTANT FOR A SPECIAL-PURPOSE DATAFLOW 2/2
LANGUAGE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB
OH SCHOOL OF ENGINEERING A J BLACK DEC 85
UNCLASSIFIED AFIT/GCS/ENG/85D-2 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
Member slot: INITIAL-SUB-ELEMENT
  Valueclass: (LIST)
  Values: NIL

Member slot: INSTANTIATE
  Valueclass: NIL
  Values: (LAMBDA (SELF)
        (LET ((NEWUNIT (CREATE.UNIT (GENSYM) NIL NIL SELF)))
          (DOLIST (SUB (GET.VALUES SELF 'INITIAL-SUB-ELEMENT))
                  (LET* ((FIRSTNAME (FIRST SUB))
                         (CLASS (SECOND SUB))
                         (SUBELEMENT (UNITMSG CLASS 'INSTANTIATE))
                         (SUB-POINTER (THIRD SUB)))
                    (ADD.VALUE NEWUNIT SUB-POINTER SUBELEMENT)
                    (PUT.VALUE SUBELEMENT 'SUPER-ELEMENT NEWUNIT)
                    (UNITMSG SUBELEMENT 'NAME-ELEMENT FIRSTNAME)))
          NEWUNIT
          ))

Member slot: LABEL
  Valueclass: (STRING)
  Values: (" ")

Member slot: POSITION
  Valueclass: NIL
  Values: NIL

Member slot: SUB-ELEMENT
  Valueclass: (GRAPH-ELEMENT)
  Values: NIL

Member slot: SUB-INPUT
  Valueclass: (JUNCTION)
  Values: NIL

Member slot: SUB-OUTPUT
  Valueclass: (JUNCTION)
  Values: NIL
```

```
    Member slot: SUB-POINTERS
      Valueclass: NIL
      Values: (SUB-ELEMENT SUB-INPUT SUB-OUTPUT)


  Unit: OUT-CONNECTOR
    Members: NIL
    Subclasses: NIL

    Member slot: CONNECT-POINT-OFFSET
      Valueclass: NIL
      Values: ((3 6))

    Member slot: DEFAULT-NAME-PREFIX
      Valueclass: NIL
      Values: (O)

    Member slot: INPUT-P
      Valueclass: NIL
      Values: (NIL)

    Member slot: LABEL
      Valueclass: NIL
      Values: ("out")

    Member slot: LABEL-OFFSET
      Valueclass: NIL
      Values: ((6 2))

    Member slot: PARENT-SUB-POINTER-SLOT
      Valueclass: NIL
      Values: (SUB-OUTPUT)

    Member slot: SHAPE
      Valueclass: NIL
      Values: ((Ø Ø) (3Ø Ø) (3Ø 12) (Ø 12) (3 6) (Ø Ø))


  Unit: OUT-PIN
    Members: NIL
    Subclasses: (#Unit (FGN-OUT-PIN FGN-GRAMMAR-KB))
```

```
  Member slot: DEFAULT-NAME-PREFIX
    Valueclass: NIL
    Values: (O)

  Member slot: OUT-ARC
    Valueclass: (ARC)
    Values: NIL

  Member slot: PARENT-SUB-POINTER-SLOT
    Valueclass: NIL
    Values: (SUB-OUTPUT)

  Member slot: SELECTABLE-ITEM-TYPE
    Valueclass: NIL
    Values: (:OUTPUT-ITEM)


Unit: PLAN
  Members: NIL
  Subclasses: NIL

  Member slot: DEFAULT-NAME-PREFIX
    Valueclass: NIL
    Values: (PLAN)

  Member slot: NIL
    Valueclass: NIL
    Values: NIL


Unit: PLAN-LIBRARY
  Members: NIL
  Subclasses: NIL


Unit: PRIMITIVE
  Members: NIL
  Subclasses: (#Unit (ROBOT-ARM FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (OFFBUTTONLIGHTS FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (HOSTOUT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (HCPIP FGN-PRIMITIVE-LIBRARY-KB)
```

```
#Unit (FLABEL0 FGN-PRIMITIVE-LIBRARY-KB)
#Unit (FLABEL.1-12 FGN-PRIMITIVE-LIBRARY-KB)
#Unit (FKEYS FGN-PRIMITIVE-LIBRARY-KB)
#Unit (FFPLOT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.ZVECTOR FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.ZROTATE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.YVECTOR FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.YROTATE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.XVECTOR FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.XROTATE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.XORC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.XOR FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.XFORMDATA FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.WINDOW FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.VECC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.VEC.EXTRACT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.VEC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.TRANS.STRING FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.TIMEOUT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.TAKE.STRING FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.SYNC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.SUBC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.SUB FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.STRING.TO.NUM FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.SQROOT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.SPLIT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.SINCOS FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.SCALE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.ROUTEC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.ROUTE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.ROUND FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.RANGE.SELECT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.PUT.STRING FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.PRINT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.POSITION.LINE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.PICKINFO FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.PASSTHRU FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.PARTS FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.ORC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.OR FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.NOT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.NOP FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.NEC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.NE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.MULC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.MUL FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.MODC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.MOD FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.MCONCATENATE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.MATRIX4 FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.MATRIX3 FGN-PRIMITIVE-LIBRARY-KB)
```

```
#Unit (F.MATRIX2 FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.LTC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.LT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.LOOKFROM FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.LOOKAT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.LINEEDITOR FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.LIMIT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.LENGHT.STRING FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.LEC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.LE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.LBL.EXTRACT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.LABEL FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.INPUTS.CHOOSE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.GTC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.GT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.GEC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.GE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.GCE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.GATHER.STRING FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.FOV FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.FLOAT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.FIX FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.FINC.STRING FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.FETCH FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.EQC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.EQ FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.EDGE.DETECT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.DZROTATE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.DYROTATE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.DXROTATE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.DSCALE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.DIV FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.DIFC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.DELETA FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CVEC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CSUB FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CSCALE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CROUTE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CROTATE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CONSTANT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CONCATENTATEC FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CONCATENATE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.COMP.STRING FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.COLOR FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CMUL FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CLTICKS FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CLT FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CLFRAMES FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CLE FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CLCSECONDS FGN-PRIMITIVE-LIBRARY-KB)
#Unit (F.CHARMASK FGN-PRIMITIVE-LIBRARY-KB)
```

```
    #Unit (F.CHARCONVERT FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.CGT FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.CELING FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.CDIV FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.CCONCATENATE FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.CBROUTE FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.BROUTEC FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.BROUTE FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.BOOLEAN.CHOOSE FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.AVERAGE FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.ATSCALE FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.AND FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.ADDC FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.ADD FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (F.ACCUMULATE FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (DTREE FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (DLABEL.1-8 FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (DIALS FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (CURSOR FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (CLEAR.LABELS FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (BUTTONSIN FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (TABLETOUT FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (TABLETIN FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (SPECKEYS FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (PICK.LOCATION FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (PICK FGN-PRIMITIVE-LIBRARY-KB)
    #Unit (KEYBOARD FGN-PRIMITIVE-LIBRARY-KB))

Member slot: DEFAULT-NAME-PREFIX
  Valueclass: NIL
  Values: (P)

Member slot: DEFINE-PRIMITIVE
  Valueclass: (METHOD)
  Values: NIL

Member slot: LABEL
  Valueclass: NIL
  Values: ("duamy")

Member slot: PARENT-SUB-POINTER-SLOT
  Valueclass: NIL
  Values: (SUB-PRIMITIVE)
```

```
  Member slot: SELECTABLE-ITEM-TYPE
    Valueclass: NIL
    Values: (:PRIMITIVE-ITEM)


Unit: PRIMITIVE-LIBRARY
  Members: NIL
  Subclasses: (#Unit (FGN-PRIMITIVE-LIBRARY FGN-PRIMITIVE-LIBRARY-KB))


Unit: UPDATE-SUB-ELEMENT-POINTER
  Members: NIL
  Subclasses: NIL

  Own slot: AVPUT
    Valueclass: NIL
    Values: (LAMBDA (SELF SLOT NEWVALUE OLDVALUE UNIT SLOTTYPE)
          (DOLIST (NEW NEWVALUE)
                  (IF (NOT (MEMBER NEW OLDVALUE)) (ADD.VALUE UNIT 'SUB-ELEMENT NEW)))
          (DOLIST (OLD OLDVALUE)
                  (IF (NOT (MEMBER OLD NEWVALUE))
                      (REMOVE.VALUE UNIT 'SUB-ELEMENT OLD)))
          NEWVALUE)
```

Knowledge Base: FGN-GRAMMAR-KB

Contents:

ANY-TYPE-INPUT
BOOLEAN-INPUT
BOOLEAN-OUTPUT
C-BOOLEAN-INPUT
CONSTANT-INPUT
FGN-DATA-TOKEN-TYPE
FGN-GRAMMAR
FGN-IN-PIN
FGN-OUT-PIN
INPUT-DETERMINES-OUTPUT
MATRIX
T.2D
T.2X2
T.3D
T.3X3
T.4D
T.4X3
T.4X4
T.BOOLEAN
T.CHARACTER
T.INTEGER
T.REAL
T.STRING
VECTOR

```
Unit: ANY-TYPE-INPUT
  Members: NIL
  Subclasses: NIL


Unit: BOOLEAN-INPUT
  Members: NIL
  Subclasses: NIL

  Member slot: LEGAL-DATA-TOKENS
    Valueclass: NIL
    Values: (T.BOOLEAN)


Unit: BOOLEAN-OUTPUT
  Members: NIL
  Subclasses: NIL

  Member slot: LEGAL-DATA-TOKENS
    Valueclass: NIL
    Values: (T.BOOLEAN)


Unit: C-BOOLEAN-INPUT
  Members: NIL
  Subclasses: NIL


Unit: CONSTANT-INPUT
  Members: NIL
  Subclasses: (#Unit (C-BOOLEAN-INPUT FGN-GRAMMAR-KB))


Unit: FGN-DATA-TOKEN-TYPE
  Members: (#Unit (T.STRING FGN-GRAMMAR-KB) #Unit (T.REAL FGN-GRAMMAR-KB)
                                    #Unit (T.INTEGER FGN-GRAMMAR-KB)
                                    #Unit (T.CHARACTER FGN-GRAMMAR-KB)
                                    #Unit (T.BOOLEAN FGN-GRAMMAR-KB))
  Subclasses: (#Unit (VECTOR FGN-GRAMMAR-KB) #Unit (MATRIX FGN-GRAMMAR-KB))


Unit: FGN-GRAMMAR
  Members: NIL
  Subclasses: NIL
```

```
  Own slot: PRIMITIVE-LIBRARY
    Valueclass: NIL
    Values: (FGN-PRIMITIVE-LIBRARY)


Unit: FGN-IN-PIN
  Members: NIL
  Subclasses: (#Unit (CONSTANT-INPUT FGN-GRAMMAR-KB) #Unit (BOOLEAN-INPUT FGN-GRAMMAR-KB)
                                          #Unit (ANY-TYPE-INPUT FGN-GRAMMAR-KB))


Unit: FGN-OUT-PIN
  Members: NIL
  Subclasses: (#Unit (BOOLEAN-OUTPUT FGN-GRAMMAR-KB)
     #Unit (INPUT-DETERMINES-OUTPUT FGN-GRAMMAR-KB))


Unit: INPUT-DETERMINES-OUTPUT
  Members: NIL
  Subclasses: NIL


Unit: MATRIX
  Members: (#Unit (T.4X4 FGN-GRAMMAR-KB) #Unit (T.4X3 FGN-GRAMMAR-KB)
                                  #Unit (T.3X3 FGN-GRAMMAR-KB)
                                  #Unit (T.2X2 FGN-GRAMMAR-KB))
  Subclasses: NIL


Unit: T.2D
  Members: NIL
  Subclasses: NIL


Unit: T.2X2
  Members: NIL
  Subclasses: NIL


Unit: T.3D
  Members: NIL
  Subclasses: NIL
```

```
Unit: T.3X3
  Members: NIL
  Subclasses: NIL

Unit: T.4D
  Members: NIL
  Subclasses: NIL

Unit: T.4X3
  Members: NIL
  Subclasses: NIL

Unit: T.4X4
  Members: NIL
  Subclasses: NIL

Unit: T.BOOLEAN
  Members: NIL
  Subclasses: NIL

Unit: T.CHARACTER
  Members: NIL
  Subclasses: NIL

Unit: T.INTEGER
  Members: NIL
  Subclasses: NIL

Unit: T.REAL
  Members: NIL
  Subclasses: NIL
```

```
Unit: T.STRING
  Members: NIL
  Subclasses: NIL


Unit: VECTOR
  Members: (#Unit (T.4D FGN-GRAMMAR-KB) #Unit (T.3D FGN-GRAMMAR-KB)
                                  #Unit (T.2D FGN-GRAMMAR-KB))
  Subclasses: NIL
```

Knowledge Base: FGN-PRIMITIVE-LIBRARY-KB

Contents:

ARITHMETIC-AND-LOGICAL
BUTTONSIN
CHARACTER-TRANSFORMATION
CLEAR.LABELS
COMPARISON
CURSOR
DATA-CONVERSION
DATA-SELECTION-AND-MANIPULATION
DIALS
DISPLAY-TREE
DLABEL.1-8
DTREE
F.ACCUMULATE
F.ADD
F.ADDC
F.AND
F.ATSCALE
F.AVERAGE
F.BOOLEAN.CHOOSE
F.BROUTE
F.BROUTEC
F.CBROUTE
F.CCONCATENATE
F.CDIV
F.CELING
F.CGT
F.CHARCONVERT
F.CHARMASK
F.CLCSECONDS
F.CLE
F.CLFRAMES
F.CLT
F.CLTICKS
F.CMUL
F.COLOR
F.COMP.STRING
F.CONCATENATE

F.CONCATENTATEC
F.CONSTANT
F.CROTATE
F.CROUTE
F.CSCALE
F.CSUB
F.CVEC
F.DELETA
F.DIFC
F.DIV
F.DSCALE
F.DXROTATE
F.DYROTATE
F.DZROTATE
F.EDGE.DETECT
F.EQ
F.EQC
F.FETCH
F.FINC.STRING
F.FIX
F.FLOAT
F.FOV
F.GATHER.STRING
F.GCE
F.GE
F.GEC
F.GT
F.GTC
F.INPUTS.CHOOSE
F.LABEL
F.LBL.EXTRACT
F.LE
F.LEC
F.LENGHT.STRING
F.LIMIT
F.LINEEDITOR
F.LOOKAT
F.LOOKFROM
F.LT
F.LTC
F.MATRIX2
F.MATRIX3
F.MATRIX4
F.MCONCATENATE
F.MOD
F.MODC
F.MUL
F.MULC
F.NE
F.NEC
F.NOP

F.NOT
F.OR
F.ORC
F.PARTS
F.PASSTHRU
F.PICKINFO
F.POSITION.LINE
F.PRINT
F.PUT.STRING
F.RANGE.SELECT
F.ROUND
F.ROUTE
F.ROUTEC
F.SCALE
F.SINCOS
F.SPLIT
F.SQROOT
F.STRING.TO.NUM
F.SUB
F.SUBC
F.SYNC
F.TAKE.STRING
F.TIMEOUT
F.TRANS.STRING
F.VEC
F.VEC.EXTRACT
F.VECC
F.WINDOW
F.XFORMDATA
F.XOR
F.XORC
F.XROTATE
F.XVECTOR
F.YROTATE
F.YVECTOR
F.ZROTATE
F.ZVECTOR
FFPLOT
FGN-PRIMITIVE-LIBRARY
FKEYS
FLABEL.1-12
FLABEL0
FUNCTION-NODE
HCPIP
HOSTOUT
INITIAL-STRUCTURES
INPUT-FUNCTIONS
KEYBOARD
MISCELLANEOUS-FUNCTIONS
OBJECT-TRANSFORMATION
OFFBUTTONLIGHTS

OUTPUT-FUNCTIONS
PICK
PICK.LOCATION
ROBOT-ARM
SPECKEYS
TABLETIN
TABLETOUT
TIMING
VIEW-TRANSFORMATION

```
Unit: ARITHMETIC-AND-LOGICAL
  Members: (#Unit (F.XORC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.XOR FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.SUBC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.SUB FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.SQROOT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.SINCOS FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.ROUND FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.ORC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.OR FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.NOT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.MULC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.MUL FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.MODC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.MOD FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.DIV FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.DIFC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.CSUB FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.CMUL FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.CDIV FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.AVERAGE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.AND FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.ADDC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.ADD FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.ACCUMULATE FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL


Unit: BUTTONSIN
  Members: NIL
  Subclasses: NIL
```

```
Unit: CHARACTER-TRANSFORMATION
  Members: (#Unit (F.CSCALE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.CROTATE FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL


Unit: CLEAR.LABELS
  Members: NIL
  Subclasses: NIL


Unit: COMPARISON
  Members: (#Unit (F.NEC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.NE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.LTC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.LT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.LEC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.LE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.GTC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.GT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.GEC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.GE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.GCE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.EQC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.EQ FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.COMP.STRING FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.CLT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.CLE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.CGT FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL


Unit: CURSOR
  Members: NIL
  Subclasses: NIL


Unit: DATA-CONVERSION
  Members: (#Unit (F.ZVECTOR FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.YVECTOR FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.XVECTOR FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.XFORMDATA FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.VECC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.VEC FGN-PRIMITIVE-LIBRARY-KB)
```

```
      #Unit (F.TRANS.STRING FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.STRING.TO.NUM FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.PRINT FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.PARTS FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.MATRIX4 FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.MATRIX3 FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.MATRIX2 FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.FLOAT FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.FIX FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.CVEC FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.CHARCONVERT FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.CELING FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL


Unit: DATA-SELECTION-AND-MANIPULATION
  Members: (#Unit (F.VEC.EXTRACT FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.TAKE.STRING FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.SPLIT FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.ROUTEC FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.ROUTE FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.RANGE.SELECT FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.PUT.STRING FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.PASSTHRU FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.MCONCATENATE FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.LINEEDITOR FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.LIMIT FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.LENGHT.STRING FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.LBL.EXTRACT FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.LABEL FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.INPUTS.CHOOSE FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.GATHER.STRING FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.FINC.STRING FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.DELETA FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.CROUTE FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.CONSTANT FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.CONCATENTATEC FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.CONCATENATE FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.CHARMASK FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.CCONCATENATE FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.CBROUTE FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.BROUTEC FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.BROUTE FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.BOOLEAN.CHOOSE FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (F.ATSCALE FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL
```

```
Unit: DIALS
  Members: NIL
  Subclasses: NIL


Unit: DISPLAY-TREE
  Members: (#Unit (ROBOT-ARM FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (DTREE FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL


Unit: DLABEL.1-8
  Members: NIL
  Subclasses: NIL


Unit: DTREE
  Members: NIL
  Subclasses: NIL


Unit: F.ACCUMULATE
  Members: NIL
  Subclasses: NIL


Unit: F.ADD
  Members: NIL
  Subclasses: NIL


Unit: F.ADDC
  Members: NIL
  Subclasses: NIL


Unit: F.AND
  Members: NIL
  Subclasses: NIL
```

```
Unit: F.ATSCALE
  Members: NIL
  Subclasses: NIL

Unit: F.AVERAGE
  Members: NIL
  Subclasses: NIL

Unit: F.BOOLEAN.CHOOSE
  Members: NIL
  Subclasses: NIL

Unit: F.BROUTE
  Members: NIL
  Subclasses: NIL

Unit: F.BROUTEC
  Members: NIL
  Subclasses: NIL

Unit: F.CBROUTE
  Members: NIL
  Subclasses: NIL

Unit: F.CCONCATENATE
  Members: NIL
  Subclasses: NIL

Unit: F.CDIV
  Members: NIL
  Subclasses: NIL
```

Unit: F.CELING
  Members: NIL
  Subclasses: NIL

Unit: F.CGT
  Members: NIL
  Subclasses: NIL

Unit: F.CHARCONVERT
  Members: NIL
  Subclasses: NIL

Unit: F.CHARMASK
  Members: NIL
  Subclasses: NIL

Unit: F.CLCSECONDS
  Members: NIL
  Subclasses: NIL

Unit: F.CLE
  Members: NIL
  Subclasses: NIL

Unit: F.CLFRAMES
  Members: NIL
  Subclasses: NIL

Unit: F.CLT
  Members: NIL
  Subclasses: NIL

```
Unit: F.CLTICKS
  Members: NIL
  Subclasses: NIL


Unit: F.CMUL
  Members: NIL
  Subclasses: NIL


Unit: F.COLOR
  Members: NIL
  Subclasses: NIL


Unit: F.COMP.STRING
  Members: NIL
  Subclasses: NIL


Unit: F.CONCATENATE
  Members: NIL
  Subclasses: NIL


Unit: F.CONCATENTATEC
  Members: NIL
  Subclasses: NIL


Unit: F.CONSTANT
  Members: NIL
  Subclasses: NIL

  Member slot: INITIAL-SUB-ELEMENT
    Valueclass: NIL
    Values: ((O1 #Unit (INPUT-DETERMINES-OUTPUT FGN-GRAMMAR-KB) SUB-OUTPUT)
       (I2 #Unit (ANY-TYPE-INPUT FGN-GRAMMAR-KB) SUB-INPUT)
       (I1 #Unit (ANY-TYPE-INPUT FGN-GRAMMAR-KB) SUB-INPUT))
```

```
  Member slot: LABEL
    Valueclass: NIL
    Values: ("f:constant")


Unit: F.CROTATE
  Members: NIL
  Subclasses: NIL


Unit: F.CROUTE
  Members: NIL
  Subclasses: NIL


Unit: F.CSCALE
  Members: NIL
  Subclasses: NIL


Unit: F.CSUB
  Members: NIL
  Subclasses: NIL


Unit: F.CVEC
  Members: NIL
  Subclasses: NIL


Unit: F.DELETA
  Members: NIL
  Subclasses: NIL


Unit: F.DIFC
  Members: NIL
  Subclasses: NIL
```

Unit: F.DIV
  Members: NIL
  Subclasses: NIL

Unit: F.DSCALE
  Members: NIL
  Subclasses: NIL

Unit: F.DXROTATE
  Members: NIL
  Subclasses: NIL

Unit: F.DYROTATE
  Members: NIL
  Subclasses: NIL

Unit: F.DZROTATE
  Members: NIL
  Subclasses: NIL

Unit: F.EDGE.DETECT
  Members: NIL
  Subclasses: NIL

Unit: F.EQ
  Members: NIL
  Subclasses: NIL

Unit: F.EQC
  Members: NIL
  Subclasses: NIL

```
Unit: F.FETCH
  Members: NIL
  Subclasses: NIL


Unit: F.FINC.STRING
  Members: NIL
  Subclasses: NIL


Unit: F.FIX
  Members: NIL
  Subclasses: NIL


Unit: F.FLOAT
  Members: NIL
  Subclasses: NIL


Unit: F.FOV
  Members: NIL
  Subclasses: NIL


Unit: F.GATHER.STRING
  Members: NIL
  Subclasses: NIL


Unit: F.GCE
  Members: NIL
  Subclasses: NIL


Unit: F.GE
  Members: NIL
  Subclasses: NIL
```

Unit: F.GEC
  Members: NIL
  Subclasses: NIL

Unit: F.GT
  Members: NIL
  Subclasses: NIL

Unit: F.GTC
  Members: NIL
  Subclasses: NIL

Unit: F.INPUTS.CHOOSE
  Members: NIL
  Subclasses: NIL

Unit: F.LABEL
  Members: NIL
  Subclasses: NIL

Unit: F.LBL.EXTRACT
  Members: NIL
  Subclasses: NIL

Unit: F.LE
  Members: NIL
  Subclasses: NIL

Unit: F.LEC
  Members: NIL
  Subclasses: NIL

Unit: F.LENGHT.STRING
  Members: NIL
  Subclasses: NIL

Unit: F.LIMIT
  Members: NIL
  Subclasses: NIL

Unit: F.LINEEDITOR
  Members: NIL
  Subclasses: NIL

Unit: F.LOOKAT
  Members: NIL
  Subclasses: NIL

Unit: F.LOOKFROM
  Members: NIL
  Subclasses: NIL

Unit: F.LT
  Members: NIL
  Subclasses: NIL

Unit: F.LTE
  Members: NIL
  Subclasses: NIL

Unit: F.MATRIX2
  Members: NIL
  Subclasses: NIL

Unit: F.MATRIX2
  Members: NIL
  Subclasses: NIL

Unit: F.MATRIX4
  Members: NIL
  Subclasses: NIL

Unit: F.MCONCATENATE
  Members: NIL
  Subclasses: NIL

Unit: F.MOD
  Members: NIL
  Subclasses: NIL

Unit: F.MODC
  Members: NIL
  Subclasses: NIL

Unit: F.MUL
  Members: NIL
  Subclasses: NIL

Unit: F.MULC
  Members: NIL
  Subclasses: NIL

Unit: F.NE
  Members: NIL
  Subclasses: NIL

Unit: F.NEC
  Members: NIL
  Subclasses: NIL

Unit: F.NOP
  Members: NIL
  Subclasses: NIL

Unit: F.NOT
  Members: NIL
  Subclasses: NIL

Unit: F.OR
  Members: NIL
  Subclasses: NIL

Unit: F.ORC
  Members: NIL
  Subclasses: NIL

Unit: F.PARTS
  Members: NIL
  Subclasses: NIL

Unit: F.PASSTHRU
  Members: NIL
  Subclasses: NIL

Unit: F.PICKINFO
  Members: NIL
  Subclasses: NIL

```
Unit: F.POSITION.LINE
  Members: NIL
  Subclasses: NIL


Unit: F.PRINT
  Members: NIL
  Subclasses: NIL


Unit: F.PUT.STRING
  Members: NIL
  Subclasses: NIL


Unit: F.RANGE.SELECT
  Members: NIL
  Subclasses: NIL


Unit: F.ROUND
  Members: NIL
  Subclasses: NIL


Unit: F.ROUTE
  Members: NIL
  Subclasses: NIL


Unit: F.ROUTEC
  Members: NIL
  Subclasses: NIL


Unit: F.SCALE
  Members: NIL
  Subclasses: NIL
```

Unit: F.SINCOS
  Members: NIL
  Subclasses: NIL

Unit: F.SPLIT
  Members: NIL
  Subclasses: NIL

Unit: F.SQROOT
  Members: NIL
  Subclasses: NIL

Unit: F.STRING.TO.NUM
  Members: NIL
  Subclasses: NIL

Unit: F.SUB
  Members: NIL
  Subclasses: NIL

Unit: F.SUBC
  Members: NIL
  Subclasses: NIL

Unit: F.SYNC
  Members: NIL
  Subclasses: NIL

Unit: F.TAKE.STRING
  Members: NIL
  Subclasses: NIL

```
Unit: F.TIMEOUT
  Members: NIL
  Subclasses: NIL

Unit: F.TRANS.STRING
  Members: NIL
  Subclasses: NIL

Unit: F.VEC
  Members: NIL
  Subclasses: NIL

Unit: F.VEC.EXTRACT
  Members: NIL
  Subclasses: NIL

Unit: F.VECC
  Members: NIL
  Subclasses: NIL

Unit: F.WINDOW
  Members: NIL
  Subclasses: NIL

Unit: F.XFORMDATA
  Members: NIL
  Subclasses: NIL

Unit: F.XOR
  Members: NIL
  Subclasses: NIL
```

```
  Member slot: DEFAULT-NAME-PREFIX
    Valueclass: NIL
    Values: (XOR)

  Member slot: INITIAL-SUB-ELEMENT
    Valueclass: NIL
    Values: ((O1 #Unit (BOOLEAN-OUTPUT FGN-GRAMMAR-KB) SUB-OUTPUT)
       (I2 #Unit (BOOLEAN-INPUT FGN-GRAMMAR-KB) SUB-INPUT)
       (I1 #Unit (BOOLEAN-INPUT FGN-GRAMMAR-KB) SUB-INPUT))

  Member slot: LABEL
    Valueclass: NIL
    Values: ("f:xor")


Unit: F.XORC
  Members: NIL
  Subclasses: NIL


Unit: F.XROTATE
  Members: NIL
  Subclasses: NIL


Unit: F.XVECTOR
  Members: NIL
  Subclasses: NIL


Unit: F.YROTATE
  Members: NIL
  Subclasses: NIL


Unit: F.YVECTOR
  Members: NIL
  Subclasses: NIL
```

```
Unit: F.IROTATE
  Members: NIL
  Subclasses: NIL


Unit: F.IVECTOR
  Members: NIL
  Subclasses: NIL


Unit: FFPLOT
  Members: NIL
  Subclasses: NIL


Unit: FGN-PRIMITIVE-LIBRARY
  Members: NIL
  Subclasses: (#Unit (DISPLAY-TREE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (FUNCTION-NODE FGN-PRIMITIVE-LIBRARY-KB))

  Member slot: MEMBER-MENU
    Valueclass: NIL
    Values: ((FUNCTION-NODE
        #Unit (FUNCTION-NODE FGN-PRIMITIVE-LIBRARY-KB)
        "See sub menu"
        (SUBITEMS
         (ARITHMETIC-AND-LOGICAL
          #Unit (ARITHMETIC-AND-LOGICAL FGN-PRIMITIVE-LIBRARY-KB)
          "See sub menu"
          (SUBITEMS ("F.XOR" #Unit (F.XOR FGN-PRIMITIVE-LIBRARY-KB)
                             "Instantiate F.XOR")
                    ("F.ACCUMULATE" #Unit (F.ACCUMULATE FGN-PRIMITIVE-LIBRARY-KB)
                                    "Instantiate F.ACCUMULATE")
                    ("F.ADD" #Unit (F.ADD FGN-PRIMITIVE-LIBRARY-KB)
                             "Instantiate F.ADD")
                    ("F.ADDC" #Unit (F.ADDC FGN-PRIMITIVE-LIBRARY-KB)
                              "Instantiate F.ADDC")
                    ("F.AND" #Unit (F.AND FGN-PRIMITIVE-LIBRARY-KB)
                             "Instantiate F.AND")
                    ("F.AVERAGE" #Unit (F.AVERAGE FGN-PRIMITIVE-LIBRARY-KB)
                                 "Instantiate F.AVERAGE")
                    ("F.CDIV" #Unit (F.CDIV FGN-PRIMITIVE-LIBRARY-KB)
                             "Instantiate F.CDIV")
                    ("F.CMUL" #Unit (F.CMUL FGN-PRIMITIVE-LIBRARY-KB)
```

```
                    "Instantiate F.CMUL")
                ("F.CSUB" #Unit (F.CSUB FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.CSUB")
                ("F.DIV" #Unit (F.DIV FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.DIV")
                ("F.DIFC" #Unit (F.DIFC FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.DIFC")
                ("F.MOD" #Unit (F.MOD FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.MOD")
                ("F.MODC" #Unit (F.MODC FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.MODC")
                ("F.MUL" #Unit (F.MUL FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.MUL")
                ("F.MULC" #Unit (F.MULC FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.MULC")
                ("F.NOT" #Unit (F.NOT FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.NOT")
                ("F.OR" #Unit (F.OR FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.OR")
                ("F.ORC" #Unit (F.ORC FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.ORC")
                ("F.ROUND" #Unit (F.ROUND FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.ROUND")
                ("F.SINCOS" #Unit (F.SINCOS FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.SINCOS")
                ("F.SQROOT" #Unit (F.SQROOT FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.SQROOT")
                ("F.SUB" #Unit (F.SUB FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.SUB")
                ("F.SUBC" #Unit (F.SUBC FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.SUBC")
                ("F.XORC" #Unit (F.XORC FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.XORC")))
("CHARACTER-TRANSFORMATION
 #Unit (CHARACTER-TRANSFORMATION FGN-PRIMITIVE-LIBRARY-KB)
 "See sub menu"
 (SUBITEMS ("F.CROTATE" #Unit (F.CROTATE FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.CROTATE")
           ("F.CSCALE" #Unit (F.CSCALE FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.CSCALE")))
("COMPARISON
 #Unit (COMPARISON FGN-PRIMITIVE-LIBRARY-KB)
 "See sub menu"
 (SUBITEMS ("F.GCE" #Unit (F.GCE FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.GCE")
           ("F.CGT" #Unit (F.CGT FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.CGT")
           ("F.CLE" #Unit (F.CLE FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.CLE")
           ("F.CLT" #Unit (F.CLT FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.CLT")
```

```
            ("F.COMP.STRING" #Unit (F.COMP.STRING FGN-PRIMITIVE-LIBRARY-KB)
                             "Instantiate F.COMP.STRING")
            ("F.EQ" #Unit (F.EQ FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.EQ")
            ("F.EQC" #Unit (F.EQC FGN-PRIMITIVE-LIBRARY-KB)
                     "Instantiate F.EQC")
            ("F.GE" #Unit (F.GE FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.GE")
            ("F.GEC" #Unit (F.GEC FGN-PRIMITIVE-LIBRARY-KB)
                     "Instantiate F.GEC")
            ("F.GT" #Unit (F.GT FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.GT")
            ("F.GTC" #Unit (F.GTC FGN-PRIMITIVE-LIBRARY-KB)
                     "Instantiate F.GTC")
            ("F.LE" #Unit (F.LE FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.LE")
            ("F.LEC" #Unit (F.LEC FGN-PRIMITIVE-LIBRARY-KB)
                     "Instantiate F.LEC")
            ("F.LT" #Unit (F.LT FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.LT")
            ("F.LTC" #Unit (F.LTC FGN-PRIMITIVE-LIBRARY-KB)
                     "Instantiate F.LTC")
            ("F.NE" #Unit (F.NE FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.NE")
            ("F.NEC" #Unit (F.NEC FGN-PRIMITIVE-LIBRARY-KB)
                     "Instantiate F.NEC")))
(DATA-CONVERSION
 #Unit (DATA-CONVERSION FGN-PRIMITIVE-LIBRARY-KB)
 "See sub menu"
 (SUBITEMS
  ("F.CELING" #Unit (F.CELING FGN-PRIMITIVE-LIBRARY-KB)
              "Instantiate F.CELING")
  ("F.CHARCONVERT" #Unit (F.CHARCONVERT FGN-PRIMITIVE-LIBRARY-KB)
                   "Instantiate F.CHARCONVERT")
  ("F.CVEC" #Unit (F.CVEC FGN-PRIMITIVE-LIBRARY-KB) "Instantiate F.CVEC")
  ("F.FIX" #Unit (F.FIX FGN-PRIMITIVE-LIBRARY-KB) "Instantiate F.FIX")
  ("F.FLOAT" #Unit (F.FLOAT FGN-PRIMITIVE-LIBRARY-KB)
             "Instantiate F.FLOAT")
  ("F.MATRIX2" #Unit (F.MATRIX2 FGN-PRIMITIVE-LIBRARY-KB)
               "Instantiate F.MATRIX2")
  ("F.MATRIX3" #Unit (F.MATRIX3 FGN-PRIMITIVE-LIBRARY-KB)
               "Instantiate F.MATRIX3")
  ("F.MATRIX4" #Unit (F.MATRIX4 FGN-PRIMITIVE-LIBRARY-KB)
               "Instantiate F.MATRIX4")
  ("F.PARTS" #Unit (F.PARTS FGN-PRIMITIVE-LIBRARY-KB)
             "Instantiate F.PARTS")
  ("F.PRINT" #Unit (F.PRINT FGN-PRIMITIVE-LIBRARY-KB)
             "Instantiate F.PRINT")
  ("F.STRING.TO.NUM" #Unit (F.STRING.TO.NUM FGN-PRIMITIVE-LIBRARY-KB)
                     "Instantiate F.STRING.TO.NUM")
  ("F.TRANS.STRING" #Unit (F.TRANS.STRING FGN-PRIMITIVE-LIBRARY-KB)
```

```
                "Instantiate F.TRANS.STRING")
 ("F.VEC" #Unit (F.VEC FGN-PRIMITIVE-LIBRARY-KB) "Instantiate F.VEC")
 ("F.VECC" #Unit (F.VECC FGN-PRIMITIVE-LIBRARY-KB) "Instantiate F.VECC")
 ("F.XFORMDATA" #Unit (F.XFORMDATA FGN-PRIMITIVE-LIBRARY-KB)
                "Instantiate F.XFORMDATA")
 ("F.XVECTOR" #Unit (F.XVECTOR FGN-PRIMITIVE-LIBRARY-KB)
              "Instantiate F.XVECTOR")
 ("F.YVECTOR" #Unit (F.YVECTOR FGN-PRIMITIVE-LIBRARY-KB)
              "Instantiate F.YVECTOR")
 ("F.ZVECTOR" #Unit (F.ZVECTOR FGN-PRIMITIVE-LIBRARY-KB)
              "Instantiate F.ZVECTOR")))
(DATA-SELECTION-AND-MANIPULATION
#Unit (DATA-SELECTION-AND-MANIPULATION FGN-PRIMITIVE-LIBRARY-KB)
"See sub menu"
(SUBITEMS
 ("F.INPUTS.CHOOSE" #Unit (F.INPUTS.CHOOSE FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.INPUTS.CHOOSE")
 ("F.ATSCALE" #Unit (F.ATSCALE FGN-PRIMITIVE-LIBRARY-KB)
              "Instantiate F.ATSCALE")
 ("F.BOOLEAN.CHOOSE" #Unit (F.BOOLEAN.CHOOSE FGN-PRIMITIVE-LIBRARY-KB)
                     "Instantiate F.BOOLEAN.CHOOSE")
 ("F.BROUTE" #Unit (F.BROUTE FGN-PRIMITIVE-LIBRARY-KB)
             "Instantiate F.BROUTE")
 ("F.BROUTEC" #Unit (F.BROUTEC FGN-PRIMITIVE-LIBRARY-KB)
              "Instantiate F.BROUTEC")
 ("F.CBROUTE" #Unit (F.CBROUTE FGN-PRIMITIVE-LIBRARY-KB)
              "Instantiate F.CBROUTE")
 ("F.CCONCATENATE" #Unit (F.CCONCATENATE FGN-PRIMITIVE-LIBRARY-KB)
                   "Instantiate F.CCONCATENATE")
 ("F.CHARMASK" #Unit (F.CHARMASK FGN-PRIMITIVE-LIBRARY-KB)
               "Instantiate F.CHARMASK")
 ("F.CONCATENATE" #Unit (F.CONCATENATE FGN-PRIMITIVE-LIBRARY-KB)
                  "Instantiate F.CONCATENATE")
 ("F.CONCATENTATEC" #Unit (F.CONCATENTATEC FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.CONCATENTATEC")
 ("F.CONSTANT" #Unit (F.CONSTANT FGN-PRIMITIVE-LIBRARY-KB)
               "Instantiate F.CONSTANT")
 ("F.CROUTE" #Unit (F.CROUTE FGN-PRIMITIVE-LIBRARY-KB)
             "Instantiate F.CROUTE")
 ("F.DELETA" #Unit (F.DELETA FGN-PRIMITIVE-LIBRARY-KB)
             "Instantiate F.DELETA")
 ("F.FINC.STRING" #Unit (F.FINC.STRING FGN-PRIMITIVE-LIBRARY-KB)
                  "Instantiate F.FINC.STRING")
 ("F.GATHER.STRING" #Unit (F.GATHER.STRING FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.GATHER.STRING")
 ("F.LABEL" #Unit (F.LABEL FGN-PRIMITIVE-LIBRARY-KB)
            "Instantiate F.LABEL")
 ("F.LBL.EXTRACT" #Unit (F.LBL.EXTRACT FGN-PRIMITIVE-LIBRARY-KB)
                  "Instantiate F.LBL.EXTRACT")
 ("F.LENGHT.STRING" #Unit (F.LENGHT.STRING FGN-PRIMITIVE-LIBRARY-KB)
                    "Instantiate F.LENGHT.STRING")
```

```
  ("F.LIMIT" #Unit (F.LIMIT FGN-PRIMITIVE-LIBRARY-kB)
            "Instantiate F.LIMIT")
  ("F.LINEEDITOR" #Unit (F.LINEEDITOR FGN-PRIMITIVE-LIBRARY-kB)
                  "Instantiate F.LINEEDITOR")
  ("F.MCONCATENATE" #Unit (F.MCONCATENATE FGN-PRIMITIVE-LIBRARY-kB)
                   "Instantiate F.MCONCATENATE")
  ("F.PASSTHRU" #Unit (F.PASSTHRU FGN-PRIMITIVE-LIBRARY-kB)
                "Instantiate F.PASSTHRU")
  ("F.PUT.STRING" #Unit (F.PUT.STRING FGN-PRIMITIVE-LIBRARY-kB)
                  "Instantiate F.PUT.STRING")
  ("F.RANGE.SELECT" #Unit (F.RANGE.SELECT FGN-PRIMITIVE-LIBRARY-kB)
                   "Instantiate F.RANGE.SELECT")
  ("F.ROUTE" #Unit (F.ROUTE FGN-PRIMITIVE-LIBRARY-kB)
            "Instantiate F.ROUTE")
  ("F.ROUTEC" #Unit (F.ROUTEC FGN-PRIMITIVE-LIBRARY-kB)
             "Instantiate F.ROUTEC")
  ("F.SPLIT" #Unit (F.SPLIT FGN-PRIMITIVE-LIBRARY-kB)
            "Instantiate F.SPLIT")
  ("F.TAKE.STRING" #Unit (F.TAKE.STRING FGN-PRIMITIVE-LIBRARY-kB)
                   "Instantiate F.TAKE.STRING")
  ("F.VEC.EXTRACT" #Unit (F.VEC.EXTRACT FGN-PRIMITIVE-LIBRARY-kB)
                   "Instantiate F.VEC.EXTRACT")))
(MISCELLANEOUS-FUNCTIONS
 #Unit (MISCELLANEOUS-FUNCTIONS FGN-PRIMITIVE-LIBRARY-kB)
 "See sub menu"
 (SUBITEMS
  ("F.COLOR" #Unit (F.COLOR FGN-PRIMITIVE-LIBRARY-kB)
            "Instantiate F.COLOR")
  ("F.EDGE.DETECT" #Unit (F.EDGE.DETECT FGN-PRIMITIVE-LIBRARY-kB)
                   "Instantiate F.EDGE.DETECT")
  ("F.FETCH" #Unit (F.FETCH FGN-PRIMITIVE-LIBRARY-kB)
            "Instantiate F.FETCH")
  ("F.NOP" #Unit (F.NOP FGN-PRIMITIVE-LIBRARY-kB) "Instantiate F.NOP")
  ("F.PICKINFO" #Unit (F.PICKINFO FGN-PRIMITIVE-LIBRARY-kB)
                "Instantiate F.PICKINFO")
  ("F.POSITION.LINE" #Unit (F.POSITION.LINE FGN-PRIMITIVE-LIBRARY-kB)
                    "Instantiate F.POSITION.LINE")
  ("F.SYNC" #Unit (F.SYNC FGN-PRIMITIVE-LIBRARY-kB) "Instantiate F.SYNC")))
(OBJECT-TRANSFORMATION
 #Unit (OBJECT-TRANSFORMATION FGN-PRIMITIVE-LIBRARY-kB)
 "See sub menu"
 (SUBITEMS ("F.DSCALE" #Unit (F.DSCALE FGN-PRIMITIVE-LIBRARY-kB)
                     "Instantiate F.DSCALE")
           ("F.DXROTATE" #Unit (F.DXROTATE FGN-PRIMITIVE-LIBRARY-kB)
                        "Instantiate F.DXROTATE")
           ("F.DYROTATE" #Unit (F.DYROTATE FGN-PRIMITIVE-LIBRARY-kB)
                        "Instantiate F.DYROTATE")
           ("F.DZROTATE" #Unit (F.DZROTATE FGN-PRIMITIVE-LIBRARY-kB)
                        "Instantiate F.DZROTATE")
           ("F.XROTATE" #Unit (F.XROTATE FGN-PRIMITIVE-LIBRARY-kB)
                       "Instantiate F.XROTATE")
```

```
            ("F.YROTATE" #Unit (F.YROTATE FGN-PRIMITIVE-LIBRARY-KB)
                         "Instantiate F.YROTATE")
            ("F.ZROTATE" #Unit (F.ZROTATE FGN-PRIMITIVE-LIBRARY-KB)
                         "Instantiate F.ZROTATE")
            ("F.SCALE" #Unit (F.SCALE FGN-PRIMITIVE-LIBRARY-KB)
                       "Instantiate F.SCALE")))
 (TIMING
  #Unit (TIMING FGN-PRIMITIVE-LIBRARY-KB)
  "See sub menu"
  (SUBITEMS ("F.CLCSECONDS" #Unit (F.CLCSECONDS FGN-PRIMITIVE-LIBRARY-KB)
                            "Instantiate F.CLCSECONDS")
            ("F.CLFRAMES" #Unit (F.CLFRAMES FGN-PRIMITIVE-LIBRARY-KB)
                          "Instantiate F.CLFRAMES")
            ("F.CLTICKS" #Unit (F.CLTICKS FGN-PRIMITIVE-LIBRARY-KB)
                         "Instantiate F.CLTICKS")
            ("F.TIMEOUT" #Unit (F.TIMEOUT FGN-PRIMITIVE-LIBRARY-KB)
                         "Instantiate F.TIMEOUT")))
 (VIEW-TRANSFORMATION
  #Unit (VIEW-TRANSFORMATION FGN-PRIMITIVE-LIBRARY-KB)
  "See sub menu"
  (SUBITEMS ("F.FOV" #Unit (F.FOV FGN-PRIMITIVE-LIBRARY-KB)
                     "Instantiate F.FOV")
            ("F.LOOKAT" #Unit (F.LOOKAT FGN-PRIMITIVE-LIBRARY-KB)
                        "Instantiate F.LOOKAT")
            ("F.LOOKFROM" #Unit (F.LOOKFROM FGN-PRIMITIVE-LIBRARY-KB)
                          "Instantiate F.LOOKFROM")
            ("F.WINDOW" #Unit (F.WINDOW FGN-PRIMITIVE-LIBRARY-KB)
                        "Instantiate F.WINDOW")))
 (OUTPUT-FUNCTIONS
  #Unit (OUTPUT-FUNCTIONS FGN-PRIMITIVE-LIBRARY-KB)
  "See sub menu"
  (SUBITEMS
   ("CLEAR.LABELS" #Unit (CLEAR.LABELS FGN-PRIMITIVE-LIBRARY-KB)
                   "Instantiate CLEAR.LABELS")
   ("DLABEL.1-8" #Unit (DLABEL.1-8 FGN-PRIMITIVE-LIBRARY-KB)
                 "Instantiate DLABEL.1-8")
   ("FFPLOT" #Unit (FFPLOT FGN-PRIMITIVE-LIBRARY-KB) "Instantiate FFPLOT")
   ("FKEYS" #Unit (FKEYS FGN-PRIMITIVE-LIBRARY-KB) "Instantiate FKEYS")
   ("FLABEL0" #Unit (FLABEL0 FGN-PRIMITIVE-LIBRARY-KB)
              "Instantiate FLABEL0")
   ("FLABEL.1-12" #Unit (FLABEL.1-12 FGN-PRIMITIVE-LIBRARY-KB)
                  "Instantiate FLABEL.1-12")
   ("HCPIP" #Unit (HCPIP FGN-PRIMITIVE-LIBRARY-KB) "Instantiate HCPIP")
   ("HOSTOUT" #Unit (HOSTOUT FGN-PRIMITIVE-LIBRARY-KB)
              "Instantiate HOSTOUT")
   ("OFFBUTTONLIGHTS" #Unit (OFFBUTTONLIGHTS FGN-PRIMITIVE-LIBRARY-KB)
                      "Instantiate OFFBUTTONLIGHTS")))
 (INPUT-FUNCTIONS
  #Unit (INPUT-FUNCTIONS FGN-PRIMITIVE-LIBRARY-KB)
  "See sub menu"
  (SUBITEMS ("BUTTONSIN" #Unit (BUTTONSIN FGN-PRIMITIVE-LIBRARY-KB)
```

```
                        "Instantiate BUTTONSIN")
                ("DIALS" #Unit (DIALS FGN-PRIMITIVE-LIBRARY-KB)
                        "Instantiate DIALS")
                ("KEYBOARD" #Unit (KEYBOARD FGN-PRIMITIVE-LIBRARY-KB)
                        "Instantiate KEYBOARD")
                ("PICK" #Unit (PICK FGN-PRIMITIVE-LIBRARY-KB)
                        "Instantiate PICK")
                ("SPECKEYS" #Unit (SPECKEYS FGN-PRIMITIVE-LIBRARY-KB)
                        "Instantiate SPECKEYS")
                ("TABLETIN" #Unit (TABLETIN FGN-PRIMITIVE-LIBRARY-KB)
                        "Instantiate TABLETIN")
                ("TABLETOUT" #Unit (TABLETOUT FGN-PRIMITIVE-LIBRARY-KB)
                        "Instantiate TABLETOUT")))
        (INITIAL-STRUCTURES
         #Unit (INITIAL-STRUCTURES FGN-PRIMITIVE-LIBRARY-KB)
         "See sub menu"
         (SUBITEMS
          ("CURSOR" #Unit (CURSOR FGN-PRIMITIVE-LIBRARY-KB) "Instantiate CURSOR")
          ("PICK.LOCATION" #Unit (PICK.LOCATION FGN-PRIMITIVE-LIBRARY-KB)
                        "Instantiate PICK.LOCATION")))))
      (DISPLAY-TREE
       #Unit (DISPLAY-TREE FGN-PRIMITIVE-LIBRARY-KB)
       "See sub menu"
       (SUBITEMS ("DTREE" #Unit (DTREE FGN-PRIMITIVE-LIBRARY-KB)
                        "Instantiate DTREE")
                ("ROBOT-ARM" #Unit (ROBOT-ARM FGN-PRIMITIVE-LIBRARY-KB)
                        "Instantiate ROBOT-ARM"))))


Unit: FKEYS
  Members: NIL
  Subclasses: NIL


Unit: FLABEL.1-12
  Members: NIL
  Subclasses: NIL


Unit: FLABEL0
  Members: NIL
  Subclasses: NIL


Unit: FUNCTION-NODE
  Members: NIL
  Subclasses: (#Unit (DATA-SELECTION-AND-MANIPULATION FGN-PRIMITIVE-LIBRARY-KB)
```

```
      #Unit (DATA-CONVERSION FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (COMPARISON FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (CHARACTER-TRANSFORMATION FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (ARITHMETIC-AND-LOGICAL FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (VIEW-TRANSFORMATION FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (TIMING FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (OUTPUT-FUNCTIONS FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (OBJECT-TRANSFORMATION FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (MISCELLANEOUS-FUNCTIONS FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (INPUT-FUNCTIONS FGN-PRIMITIVE-LIBRARY-KB)
      #Unit (INITIAL-STRUCTURES FGN-PRIMITIVE-LIBRARY-KB))


Unit: HCPIP
  Members: NIL
  Subclasses: NIL


Unit: HOSTOUT
  Members: NIL
  Subclasses: NIL


Unit: INITIAL-STRUCTURES
  Members: (#Unit (CURSOR FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (PICK.LOCATION FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL


Unit: INPUT-FUNCTIONS
  Members: (#Unit (DIALS FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (BUTTONSIN FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (TABLETOUT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (TABLETIN FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (SPECKEYS FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (PICK FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (KEYBOARD FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL


Unit: KEYBOARD
  Members: NIL
  Subclasses: NIL
```

```
Unit: MISCELLANEOUS-FUNCTIONS
  Members: (#Unit (F.SYNC FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.POSITION.LINE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.PICKINFO FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.NOP FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.FETCH FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.EDGE.DETECT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.COLOR FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL


Unit: OBJECT-TRANSFORMATION
  Members: (#Unit (F.ZROTATE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.YROTATE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.XROTATE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.SCALE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.DZROTATE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.DYROTATE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.DXROTATE FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.DSCALE FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL


Unit: OFFBUTTONLIGHTS
  Members: NIL
  Subclasses: NIL


Unit: OUTPUT-FUNCTIONS
  Members: (#Unit (OFFBUTTONLIGHTS FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (HOSTOUT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (HCPIP FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (FLABEL0 FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (FLABEL.1-12 FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (FKEYS FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (FFPLOT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (DLABEL.1-8 FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (CLEAR.LABELS FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL


Unit: PICK
  Members: NIL
  Subclasses: NIL
```

```
Unit: PICK.LOCATION
  Members: NIL
  Subclasses: NIL


Unit: ROBOT-ARM
  Members: NIL
  Subclasses: NIL


Unit: SPECKEYS
  Members: NIL
  Subclasses: NIL


Unit: TABLETIN
  Members: NIL
  Subclasses: NIL


Unit: TABLETOUT
  Members: NIL
  Subclasses: NIL


Unit: TIMING
  Members: (#Unit (F.TIMEOUT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.CLTICKS FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.CLFRAMES FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.CLCSECONDS FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL


Unit: VIEW-TRANSFORMATION
  Members: (#Unit (F.WINDOW FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.LOOKFROM FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.LOOKAT FGN-PRIMITIVE-LIBRARY-KB)
     #Unit (F.FOV FGN-PRIMITIVE-LIBRARY-KB))
  Subclasses: NIL
```

Appendix B: Graph Design Assistant Source Code

The section contains listings Lisp code used in the Graph Design Assistant. There are three files. Each file starts with a file header and a listing of the file's contents.

```
;;; -*- Mode: LISP; Package: KEE; Base: 10 -*-
;
;    Filename:    gda.lisp
;    Project:     Graph Design Assistant version 8
;    Date:        December 3, 1985
;    Author:      Alan J. Black
;    Description:
;
;      Contains misc. lisp functions used by the the graph design assistant.
;
;
;    Contents:
;
;        *host-dir*              variable
;        *gda-kbs*               variable
;        gda                     functions
;        load-gda-kbs            function
;        save-gda-kbs            function
;        select-from-alist       function
;        get-confirmation        function
;

;~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
;   GDA global variables
;
;      *host-dir*  - host directory for GDA system
;      *gda-kbs*   - list of KEE databases in the GDA system
;
(defvar *host-dir* ">black>gda>")

(defvar *gda-kbs*
        '(
          gda-system
          fgn-grammar-kb
          fgn-primitive-library-kb
          ))

;~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
;    gda ()  function
;
;        Initializes GDA.  Loads files and knowledge bases.  Sets KEE variables.

(defun gda ()
  (load ">black>gda>graph-window")
  (load ">black>gda>interface-manager")
```

```
(load-gda-kbs)
(princ "Graph Design Assistant loaded")
(setq valueclasscheck 'strict))


;***********************************************************
;    load-gda-kbs
;
;    Loads the GDA knowledge bases
;
(defun load-gda-kbs ()
    (dolist (kb *gda-kbs*)
      (kbload (string-append *host-dir* (get-pname kb)))))

;***********************************************************
;    save-gda-kbs
;
;    Save the GDA knowledge bases
;
(defun save-gda-kbs ()
     (dolist (kb *gda-kbs*)
       (kbsave kb nil *host-dir*)))


;***********************************************************
;
;    select-from-alist  (alist &optional  label)
;
;        Displays a list of available units and allows the user to choose
;        from the list.
;
;        The available-units-list is an association list formated as follows:
;
;           (("label 1" . value-1)
;            ("label 2" . value-2)
;                   ...
;
(defun select-from-alist (alist &optional label)
  (declare (special menu-window menu-list))
  (setq menu-list nil)
  (setq menu-window
        (tv:make-window
          'tv:dynamic-momentary-menu
          :borders 2
          :default-font fonts:hl10
          :label label
          :deexposed-typeout-action :permit
          :item-list-pointer 'menu-list))
  (setq menu-list alist)
  (send menu-window :choose))
```

```
;*****************************************************
;   get-confirmation (prompt)
;
;   Displays a meu and asks the user to confirm a pending action
;
(defun get-confirmation (prompt)
  (declare (special confirm-window confirm-alist))
  (setq confirm-alist nil)
  (setq confirm-window
        (tv:make-window
          'tv:dynamic-momentary-menu
          :borders 2
          :default-font fonts:hl10
          :label "CONFIRM:"
          :deexposed-typeout-action :permit
          :item-list-pointer 'confirm-alist))
  (setq confirm-alist `((,prompt . ,t) ("abort" . ,nil)))
  (send confirm-window :choose))
```

```
;;; -*- Mode: LISP; Base: 10; Package: KEE -*-
;
;    File name:   graph-window.lisp
;    Project:     Graph Design Assistant, version 8
;    Date:        December 3, 1985
;    Author:      Alan J. Black
;    Description:
;
;        Contains the definition for the graph-window-flavor and its
;        associated methods.  The graph-window-flavor is the type of
;        window which is used to edit graphs in GDA.
;
;    Contents:
;
;       graph-window-flavor          flavor definition
;       typeout-window-flavor        flavor definition
;       :calc-box-size               graph-window-flavor method
;       :draw-arc                    graph-window-flavor method
;       :draw-box                    graph-window-flavor method
;       :draw-connector              graph-window-flavor method
;       :draw-line-between-points    graph-window-flavor method
;       :draw-node                   graph-window-flavor method
;           draw-io-pair             function
;           make-io-pair-list        function
;           calc-box-width           function
;           calc-box-height          function
;       :draw-selectable-string      graph-window-flavor method
;       :get-instantiate-location    graph-window-flavor method
;       :label-graph-window          graph-window-flavor method
;       :update-path                 graph-window-flavor method
;       :show-menu                   graph-window-flavor method
;       :update                      graph-window-flavor method
;       :user-message                graph-window-flavor method
;       initial-gw-item-list         function
;       make-graph-window            function
;       make-typeout-window          function
;

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;   graph-window-flavor
;
;   Flavor definition. "k:kwin" is the window flavor used by KEE, mixing in
;   gives the graph-window-flavor similar properties.
;
(defflavor graph-window-flavor ()
```

```
            (tv:basic-mouse-sensitive-items
             tv:window-with-typeout-mixin
             tv:select-mixin
             k:kwin)
   :gettable-instance-variables
   :settable-instance-variables)

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;   typeout-window-flavor
;
;   This window will allow for errors to type out
;
(defflavor typeout-window-flavor ()
   (tv:temporary-typeout-window
    tv:select-mixin))
;
;  some constants that affect the graphic figures displayed in
;  the graph-window
;
(defconst line-space 12)
(defconst v-connect-point-offset 5)
(defconst box-top-margin 2)
(defconst box-bottom-margin 2)
(defconst min-box-width 30)
(defconst min-box-height 30)

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;    :calc-box-size (node)
;
;    Calculate the size of a node box
;
(defmethod (graph-window-flavor :calc-box-size) (node)
  (let*
    ((input-units (get.values node 'sub-input))
     (output-units (get.values node 'sub-output))
     (in-firstnames  (mapcar #'(lambda (u) (get.value u 'firstname)) input-units))
     (out-firstnames (mapcar #'(lambda (u) (get.value u 'firstname)) output-units))
     (label (get.value node 'label))
     (box-width  (calc-box-width  self label  in-firstnames out-firstnames))
     (box-height (calc-box-height in-firstnames out-firstnames)))
    `(,box-width ,box-height)))

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;    :draw-arc (arc)
;
;    Draw an arc
;
(defmethod (graph-window-flavor :draw-arc) (arc)
   (let*
    ((from (get.value arc 'from-junction))
     (to   (get.value arc 'to-junction))
```

```
      (p1   (get.value from 'output-position))
      (p2   (get.value to   'input-position))
      (path (get.values arc 'path)))
     (if
      path
      (send self :draw-line-between-points `(,p1 ,@path ,p2))
      (send self :draw-line-between-points `(,p1 ,p2)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;    :draw-box  (x y height width alu)
;
;    Draw a box
;
(defmethod (graph-window-flavor :draw-box) (x y height width &optional (alu tv:alu-ior))
    (let*
     ((lf x)
      (rt (+ x width))
      (top y)
      (bot (+ y height)))
     (send self :draw-lines alu lf top rt top rt bot lf bot lf top))
    (send self :set-cursorpos x y))

(defun xpos (position) (first position))
(defun ypos (position) (second position))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;    :draw-connector (connector input-p)
;
(defmethod (graph-window-flavor :draw-connector) (connector)
  (let*
    ((pos (get.value connector 'position))
     (x (xpos pos))
     (y (ypos pos))
     (firstname (get.value connector 'firstname))
     (input-p (get.value connector 'input-p))
     (shape (get.values connector 'shape))
     (connect-point-offset (get.value connector 'connect-point-offset))
     (connect-point `(,(+ x (xpos connect-point-offset))
                      ,(+ y (ypos connect-point-offset))))
     (label (get.value connector 'label))
     (label-offset (get.value connector 'label-offset))
     (points
       (mapcar #'(lambda (point) `(,(+ x (xpos point)) ,(+ y (ypos point))))
               shape))
     (line-args `(,tv:alu-ior ,@(flatten points))))
    (if
     input-p
     (put.value connector 'output-position connect-point)
     (put.value connector 'input-position connect-point))
    (send self :set-cursorpos x (- y line-space))
    (send self :draw-selectable-string
```

```
        (format nil "~a" firstname)
        (if input-p :in-connector-item :out-connector-item) connector)
    (lexpr-send self  :draw-lines line-args)
    (send self :set-cursorpos (+ x (xpos label-offset)) (+ y (ypos label-offset)))
    (send self :string-out label)))

;*************************************************************
;    :draw-line-between-points (point-list &optional (alu tv:alu-ior))
;
(defmethod (graph-window-flavor :draw-line-between-points)
           (point-list &optional (alu tv:alu-ior))
  (let
    ((line-args `(,alu ,@(flatten-point-list point-list))))
    (lexpr-send self :draw-lines line-args)))

(defun flatten-point-list (point-list)
  (if
    point-list
    (let*
      ((p1 (car point-list)))
      (if
        p1
        (let
          ((x (first p1))
           (y  (second p1)))
          (cons x (cons y (flatten-point-list (cdr point-list)))))
        (flatten-point-list (cdr point-list))))))

;*************************************************************
;    :draw-node (node)
;
;   Draws a node with labels and io-pins
;
(defmethod (graph-window-flavor :draw-node) (node)
  (let*
    ((position (get.value node 'position))
     (x (first position))
     (y (second position))
     (input-units (get.values node 'sub-input))
     (output-units (get.values node 'sub-output))
     (in-firstnames  (mapcar #'(lambda (u) (get.value u 'firstname)) input-units))
     (out-firstnames (mapcar #'(lambda (u) (get.value u 'firstname)) output-units))
     (node-firstname (get.value node 'firstname))
     (label (get.value node 'label))
     (box-width  (calc-box-width  self label  in-firstnames out-firstnames))
     (box-height (calc-box-height in-firstnames out-firstnames))
     (;itemtype    (get.value node 'selectable-item-type)))
    (send self :draw-box x y box-height box-width)
    (send self :set-cursorpos x (- y line-space))
    (send self :draw-selectable-string
          (format nil "~a" node-firstname) itemtype node)
```

```
    (send self :set-cursorpos x (setq y (+ y box-top-margin)))
    (if label
        (let*
          ((label-length (send self :string-length label))
           (l-x (+ x (// (- box-width label-length) 2))))
          (send self :set-cursorpos l-x y)
          (send self :string-out label)
          (setq y (+ y line-space))))
    (dolist
      (iopair (make-io-pair-list input-units output-units))
      (draw-io-pair x y box-width iopair self)
      (setq y (+ y line-space)))))


(defun draw-io-pair (x y box-width io-pair window)
  (let
    ((in-unit  (car io-pair))
     (out-unit (cdr io-pair)))
    (if in-unit
        (let
          ((in-name (get.value in-unit 'firstname))
           (connect-pos '(,x ,(+ y y-connect-point-offset))))
          (put.value in-unit 'input-position connect-pos)
          (send window :set-cursorpos x y)
          (send window :draw-selectable-string
                (format nil " ~a " in-name) :input-item in-unit)))
    (if out-unit
        (let*
          ((out-name (get.value out-unit 'firstname))
           (out-string (format nil " ~a " out-name))
           (out-length (send window :string-length out-string))
           (out-x-offset (- box-width out-length))
           (connect-pos '(,(+ x box-width) ,(+ y y-connect-point-offset))))
          (put.value out-unit 'output-position connect-pos)
          (send window :set-cursorpos (+ x out-x-offset) y)
          (send window :draw-selectable-string
                out-string :output-item out-unit)))))

(defun make-io-pair-list (in-list out-list)
  (cond
    ((and (null in-list) (null out-list)) nil)
    ((null in-list)
     (cons (cons nil (car out-list)) (make-io-pair-list nil (cdr out-list))))
    ((null out-list)
     (cons (cons (car in-list) nil) (make-io-pair-list (cdr in-list) nil)))
    (t
     (cons (cons (car in-list) (car out-list))
           (make-io-pair-list (cdr in-list) (cdr out-list))))))

(defun calc-box-width (window label in-firstnames out-firstnames)
  (let*
```

```
    ((io-pairs (pairlis in-firstnames out-firstnames))
     (pair-widths
       (mapcar
         #'(lambda (io-pair)
             (let*
               ((in-name (car io-pair))
                (out-name (cdr io-pair))
                (out-width (if out-name
                               (send window :string-length (format nil " ~a " out-name))
                               0))
                (in-width  (if in-name
                               (send window :string-length (format nil " ~a " in-name))
                               0)))
               (+ out-width in-width)))
         io-pairs))
     (max-io-pair-width (if pair-widths (apply 'max pair-widths) 0))
     (label-width (send window :string-length (format nil " ~a " label))))
    (max max-io-pair-width label-width min-box-width)))

(defun calc-box-height ( in-firstnames out-firstnames)
  (let*
    ((max-names (max (length in-firstnames) (length out-firstnames))))
    (max min-box-height
         (+ box-top-margin line-space (* max-names line-space)))))

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;    :draw-selectable-string (string itemtype item)
;
;    Draws a selectable string and adds the item to the
;    item alist.
;
(defmethod (graph-window-flavor :draw-selectable-string) (string itemtype item)
  (let (x0 y0 x1 y1)
    (multiple-value (x0 y0) (send self :read-cursorpos))
    (send self :string-out string)
    (multiple-value (x1 y1) (send self :read-cursorpos))
    (send self :primitive-item itemtype item x0 y0 x1 (+ y0 10))))


;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;    :get-instantiate-location
;
;        Allows user to move point to a position in the window with the mouse.
;
;        Returns:  coordinates  (x y), or nil if user aborts
;
(defmethod (graph-window-flavor :get-location) (width height)
  (send self :expose)
  (tv:mouse-wait)
  (tv:with-mouse-grabbed-on-sheet ()
    (do
```

```
      ((x (- tv:mouse-x 8) (- tv:mouse-x 8))
       (y (- tv:mouse-y 24) (- tv:mouse-y 24))
       (b tv:mouse-last-buttons tv:mouse-last-buttons))
      ((neq b 0) (if (= b 1) (list x y) nil))
      (draw-locate-box self x y width height)
      (send self :user-message (format nil "L:Locate, R:Abort x:~4d y:~4d" x y))
      (tv:mouse-wait)
      (draw-locate-box self x y width height))))

(defun draw-locate-box (window x0 y0 width height)
     (let
      ((x1 (+ x0 width))
       (y1 (+ y0 height)))
      (send window :draw-lines tv:alu-xor x0 y0 x1 y0 x1 y1 x0 y1 x0 y0)))



;*************************************************************
;    :label-graph-window  current-graph
;
(defmethod (graph-window-flavor :label-graph-window) (current-graph)
  (let (graphname kbname)
    (cond
      (current-graph
       (setq graphname (unit.name current-graph))
       (setq kbname    (kb.name (unit.kb current-graph))))
      (t
       (setq graphname "not assigned")
       (setq kbname    "not assigned")))
  (send self :set-label
        (format nil "graph: ~a     kb: ~a" graphname kbname))))

;*************************************************************
;    :update-path  (p1 path p2)
;
(defmethod (graph-window-flavor :update-path) (p1 path p2)
  (send self :expose)
  (send self :draw-line-between-points `(,p1 ,@path ,p2) tv:alu-xor)
  (tv:with-mouse-grabbed-on-sheet ()
    (tv:mouse-wait)
    (do
      ((quit nil)
       (new-path nil)
       (x1 (first p1))
       (y1 (second p1))
       (x (- tv:mouse-x 8)  (- tv:mouse-x 8))
       (y (- tv:mouse-y 24) (- tv:mouse-y 24))
       (b tv:mouse-last-buttons tv:mouse-last-buttons))
      (quit
       (send self :draw-line-between-points `(,p1 ,@new-path ,p2))
       new-path)
```

```
(selectq b
  (1 (send self :draw-line-between-points `(,p1 ,@new-path) tv:alu-xor)
     (setq new-path (append new-path `((,x ,y))))
     (send self :draw-line-between-points `(,p1 ,@new-path) tv:alu-xor)
     (setq x1 x)
     (setq y1 y))
  (2
   (setq new-path (butlast new-path))
   (if new-path
       (let
         ((last-point (first (last new-path))))
         (send self :draw-line-between-points
               `(,last-point (,x1 ,y1)) tv:alu-xor)
         (setq x1 (first last-point))
         (setq y1 (second last-point)))
       (progn
         (send self :draw-line-between-points
               `(,p1 (,x1 ,y1)) tv:alu-xor)
         (setq x1 (first p1))
         (setq y1 (second p1)))))
  (4 (setq quit t)))
(send self :draw-line-between-points `((,x1 ,y1) (,x ,y) ,p2) tv:alu-xor)
(send self :user-message
      (format nil
              "L:add-point L:delete-last-point R:exit (~4d,~4d) (~4d,~4d)" x1 y1 x y))
(tv:mouse-wait)
(send self :draw-line-between-points `((,x1 ,y1) (,x ,y) ,p2) tv:alu-xor)
)))

;***************************************************
;    :set-cursorpos-rel
;
(defmethod (graph-window-flavor :set-cursorpos-rel) (d-x d-y)
   (let (x y)
     (multiple-value (x y) (send self :read-cursorpos))
     (send self :set-cursorpos (+ x d-x) (+ y d-y))))

;***************************************************
;    :show-menu
;
;       Writes the list of menu items across the top of the window
;
(defmethod (graph-window-flavor :show-menu) ()
  (send self :set-cursorpos 5 2)
  (mapcar
   #'(lambda (arg)
       (lexpr-send self arg)
       (send self :string-out "    "))
   '(
     (:item :menu-zoom-out       "zoom-out")
     (:item :menu-window         "window")
```

```
    (:item :menu-agenda        "agenda")
    (:item :menu-graph         "graph")
    (:item :menu-plan          "plan")
    (:item :menu-primitive     "primitive")
    (:item :menu-function      "function")
    ))

;*************************************************
;    :update graph kb
;
(defmethod (graph-window-flavor :update) (current-graph)
     (send self :clear-window)
     (send self :show-menu)
     (send self :label-graph-window current-graph))


;*************************************************
;    :user-message <message string>
;
;        Displays the message in the lower left corner of the window
;
(defmethod (graph-window-flavor :user-message) (message-string)
    (send self :home-down)
    (send self :clear-rest-of-line)
    (send self :string-out message-string))

;*************************************************
;   initial-gw-item-alist
;
;   This function initializes the item alist for the graph-window
;
(defun initial-gw-item-alist ()
  (let ((gw-item-alist nil))
    (tv:add-typeout-item-type
      gw-item-alist
      :menu-zoom-out
      "zoom-out"
      :zoom-out
      t
      "Zoom out and view parent of this graph")

;                        ---- menu-window items ----

    (tv:add-typeout-item-type
      gw-item-alist
      :menu-window
      "assign-graph"
      :assign-graph
      nil
      "Assigns a graph to this window")
```

```
(tv:add-typeout-item-type
  gw-item-alist
  :menu-window
  "update-window"
  :update-window
  t
  "Redraw the window")

;                          ---- menu-agenda items ----

(tv:add-typeout-item-type
  gw-item-alist
  :menu-agenda
  "describe-agenda-item"
  :describe-agenda-item
  nil
  "Shows a description of an item that is on the agenda")

(tv:add-typeout-item-type
  gw-item-alist
  :menu-agenda
  "delete-agenda-item"
  :delete-agenda-item
  nil
  "Deletes an item from the agenda without executing it")

(tv:add-typeout-item-type
  gw-item-alist
  :menu-agenda
  "select-agenda-item"
  :select-agenda-item
  t
  "Selects an item from the agenda to execute")

;                          ---- menu-graph items ----


(tv:add-typeout-item-type
  gw-item-alist
  :menu-graph
  "create-graph"
  :create-graph
  nil
  "Create a graph, independent of the graph assigned to this window")

(tv:add-typeout-item-type
  gw-item-alist
  :menu-graph
  "create-sub-graph"
  :create-sub-graph
  t
```

```
    "Create a new graph as a child of the graph assigned to this window")


;                          ---- menu-plan items ----


  (tv:add-typeout-item-type
    gw-item-alist
    :menu-plan
    "describe-plan"
    :describe-plan
    nil
    "Shows a description of a plan in the plan library")

  (tv:add-typeout-item-type
    gw-item-alist
    :menu-plan
    "instantiate-plan"
    :instantiate-plan
    t
    "Gets a plan from the library and instantiates it as the child of the assigned graph")


;                          ---- menu-primitive items ----

  (tv:add-typeout-item-type
    gw-item-alist
    :menu-primitive
    "describe-primitive"
    :describe-primitive
    nil
    "Shows a description of a primitive in the primitive library")

  (tv:add-typeout-item-type
    gw-item-alist
    :menu-primitive
    "instantiate-primitive"
    :instantiate-primitive
    t
    "Gets a library primitive and instantiaties as the child of the assigned graph")

;                          ---- menu-junction ----


  (tv:add-typeout-item-type
    gw-item-alist
    :menu-junction
    "instantiate-junction"
    :instantiate-junction
    t
```

```
    "Instantiate a junction for external connections")


;                          ---- graph-item ---
(tv:add-typeout-item-type
  gw-item-alist
  :graph-item
  "label-graph"
  :label-graph
  nil
  "Assign a label to this graph (displayed inside box)")

(tv:add-typeout-item-type
  gw-item-alist
  :graph-item
  "rename-graph"
  :rename-element
  nil
  "Renames this graph node")

(tv:add-typeout-item-type
  gw-item-alist
  :graph-item
  "delete-graph"
  :delete-element
  nil
  "Deletes this graph node and all it's children")

(tv:add-typeout-item-type
  gw-item-alist
  :graph-item
  "zoom-in"
  :zoom-in
  t
  "Zoom in on this graph node")



;                          ---- primitive-item ---

(tv:add-typeout-item-type
  gw-item-alist
  :primitive-item
  "delete-primitive"
  :delete-element
  nil
  "Delete this primitive node")

(tv:add-typeout-item-type
  gw-item-alist
  :primitive-item
```

```
    "rename-primitive"
    :rename-element
    nil
    "Rename this primitive node")

;                          ---- in-connector-item ---

(tv:add-typeout-item-type
  gw-item-alist
  :in-connector-item
  "rename-out-connector"
  :rename-element
  nil
  "Rename this in-connector")

(tv:add-typeout-item-type
  gw-item-alist
  :in-connector-item
  "delete-connector"
  :delete-element
  nil
  "Delete connection node")

(tv:add-typeout-item-type
  gw-item-alist
  :input-item
  "delete-out-arc"
  :delete-out-arc
  nil
  "Delete an out arc")

(tv:add-typeout-item-type
  gw-item-alist
  :in-connector-item
  "modify-out-arc-path"
  :modify-out-arc-path
  nil
  "Change the path of the arc")

(tv:add-typeout-item-type
  gw-item-alist
  :in-connector-item
  "arc-start"
  :arc-start
  t
  "Start a connection arc")


;                          ---- out-connector-item ---

(tv:add-typeout-item-type
```

```
    gw-item-alist
    :out-connector-item
    "rename-out-connector"
    :rename-element
    nil
    "Rename this out-connector")

  (tv:add-typeout-item-type
    gw-item-alist
    :out-connector-item
    "delete-connector"
    :delete-element
    nil
    "Delete connection node")

  (tv:add-typeout-item-type
    gw-item-alist
    :out-connector-item
    "arc-complete"
    :arc-complete
    t
    "Start a connection arc")


;                          ---- input-item ---

  (tv:add-typeout-item-type
    gw-item-alist
    :input-item
    "rename-input"
    :rename-element
    nil
    "Rename this input")


  (tv:add-typeout-item-type
    gw-item-alist
    :input-item
    "arc-complete"
    :arc-complete
    t
    "Complete a connection arc")

;                          ---- output-item ---

  (tv:add-typeout-item-type
    gw-item-alist
    :output-item
    "rename-output"
    :rename-element
    nil
```

```
        "Rename this output")


      (tv:add-typeout-item-type
        gw-item-alist
        :output-item
        "delete-out-arc"
        :delete-out-arc
        nil
        "Delete an in arc")

      (tv:add-typeout-item-type
        gw-item-alist
        :output-item
        "modify-out-arc-path"
        :modify-out-arc-path
        nil
        "Change the path of the arc")

      (tv:add-typeout-item-type
        gw-item-alist
        :output-item
        "arc-start"
        :arc-start
        t
        "Start a connection arc")


;                          ---- end of typeout item types ----
    gw-item-alist))

;*****************************************************************
;    make-graph-window
;
;       Creates a new graph window an starts a graph window process.
;
;           returns:  zeta window object
;
(defun make-graph-window (item-alist)
  (tv:make-window
    'graph-window-flavor
    :label "Graph Window"
    :edges-from :mouse
    :expose-p t
    :save-bits t
    :blinker-p nil
    :item-type-alist item-alist))

;*****************************************************************
;  make-typeout-window
;
```

```
(defun make-typeout-window (superior-window)
    (tv:make-window
      'typeout-window-flavor
      :superior superior-window))


(compile-flavor-methods graph-window-flavor)
(compile-flavor-methods typeout-window-flavor)
```

```
;;; -*- Mode: LISP; Base: 10; Package: KEE -*-
;
;    File name:   interface-manager.lisp
;    Project:     Graph Design Assistant, version 8
;    Date:        December 3, 1985
;    Author:      Alan J. Black
;    Description:
;
;        Containts the mouse handler process which defines the user interface
;        to a graph-window.
;
;    Contents:
;
;         start-graph-window-process      function
;         graph-window-process            function
;         check-for-terminal-io           function
;
;      Functions that execute user commands:
;
;         arc-complete                    function
;         arc-start                       function
;         assign-graph                    function
;         create-graph                    function
;         create-sub-graph                function
;         delete-element                  function
;         delete-out-arc                  function
;         instantiate-junction            function
;         instantiate-primitive           function
;         modify-out-arc-path             function
;         rename-element                  function
;         update-graph-window             function
;         zoom-in                         function
;         zoom-out                        function
;

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;   start-graph-window-process
;
;   Starts a graph window process
;
(defun start-graph-window-process ()
  (process-run-function '(:name "graph-window-process")
    #'graph-window-process
    ))
```

```
(defvar *gw*)  ; Contains the current graph-window object, for debugging purposes

;;; ------------------------------------------------------------------------
;    graph-window-process
;
;    The process which handles mouse inputs for a window.
;
;        The function is a loop which reads in a mouse "blip",
;        decodes it, then calls the function requested by the user.
;
;        The loop is endless, however, the user can kill the process by
;        closing the window (a KEE window property).
;
(defun graph-window-process ()
  (let*
    ((graph nil)
     (current-out-junction nil)
     (item-alist (initial-gw-item-alist))
     (window     (make-graph-window item-alist))
     (terminal-io (make-typeout-window window)))
    (tv:window-mouse-call (window :deactivate)
      (setq *gw* window)                        ; for debugging
      (pkg-goto 'kee)
      (update-graph-window window graph)
      (error-restart-loop ((sys:abort) "Restart graph-window-process" nil)
        (check-for-terminal-io terminal-io)
        (let*
          ((blip (send window :any-tyi))
           (command (second blip))
           (object  (third  blip)))
          (send window :user-message " ")
          (if (and current-out-junction (neq command :arc-complete))
              (setq current-out-junction nil))
          (selectq command
            (:arc-complete
             (setq current-out-junction
                   (arc-complete window graph object current-out-junction)))
            (:arc-start (setq current-out-junction (arc-start window graph object)))
            (:update-window
             (update-graph-window window graph))
            (:assign-graph
             (setq graph (assign-graph window graph)))
            (:create-graph
             (setq graph (create-graph window graph)))
            (:create-sub-graph
             (create-sub-graph window graph))
            (:delete-element (delete-element window graph object))
            (:delete-out-arc
             (delete-out-arc window graph object))
            (:instantiate-junction  (instantiate-junction window graph))
            (:instantiate-primitive (instantiate-primitive window graph))
```

```
          (:modify-out-arc-path (modify-out-arc-path window object))
          (:rename-element (rename-element window object graph))
          (:zoom-in (setq graph (zoom-in window object)))
          (:zoom-out (setq graph (zoom-out window graph)))
          (otherwise
           (send window :user-message
                (format nil "Command not implemented: ~s" blip))))))))

;*************************************************
;
;    check-current-graph (window graph)
;
;    Check to see if graph unit exists. if not display message in
;    window and return nil
;
(defun check-current-graph (window graph)
   (cond
     ((null graph)
      (send window :user-message (format nil "A graph is not assigned to this window")))
     ((null (unitreference* graph))
      (send window :user-message (format nil "errror: ~s is not a graph?" graph)))
     (t graph)))

;*************************************************
;    Check to see if any type out has occurred on the typeout window.
;    if it has. wait until the user is ready to continue.
;
(defun check-for-terminal-io (tio)
   (cond
     ((send tio :exposed-p)
      (send tio :string-out (format nil "~%click mouse on window to continue"))
      (send tio :any-tyi)
      (send tio :deexpose))))


;*************************************************
;    arc-complete (window graph in-junction current-out-junction)
;
;    Completes an arc.
;
;       Creates an ARC element and places pointers in
;       it's FROM-JUNCTION and TO-JUNCTION slot of the arc unit.
;
;       Places pointers in the OUT-ARC slot of the out junction, and the IN-ARC
;       slot of the in junction.
;
(defun arc-complete (window graph in-junction current-out-junction)
  (if
    (null current-out-junction)
    (send window :user-message "Start arc on an output of a node")
    (let
```

```
   ((arc (unitmsg graph 'add-sub-element '(arc gda-system))))
   (put.value arc 'from-junction current-out-junction)
   (put.value arc 'to-junction   in-junction)
   (add.value current-out-junction 'out-arc arc)
   (add.value in-junction 'in-arc arc)
   (update-graph-window window graph)))
 nil)

;************************************************************
;   arc-start (window graph out-junction)
;
;   Sends the user a message that he has started an arc.
;
;       Returns the out junction.
;
;
(defun arc-start (window graph out-junction)
  (let*
    ((out-pin-name (get.value out-junction 'firstname))
     (parent-node (get.value out-junction 'super-element))
     (node-name (get.value parent-node 'firstname)))
    (send window :user-message
         (format nil "Arc started from ~a, ~a" out-pin-name node-name))
    out-junction))

;************************************************************
;   assign-graph  (window current-graph)
;
;   Assigns the graph that is to be displayed in the graph-window.
;
;      The user is presented with a pop-up menu where he can fill in
;      the name of the graph and the knowledge base that contains the graph.
;
;      Returns the unit.reference to the graph unless there is an error, then
;      nil is returned.
;
(defun assign-graph (window current-graph)
  (declare (special graph-name kb-name))
  (cond
    ((null current-graph)
     (setq graph-name nil)
     (setq kb-name nil))
    ((unitreference* current-graph)
     (setq graph-name (unit.name current-graph))
     (setq kb-name    (kb.name (unit.kb current-graph)))))
  (tv:choose-variable-values
    '((graph-name "Graph          ")
      (kb-name    "Knowledge base"))
    :label "Assign graph to window     ")
  (cond
    ((setq current-graph (unitreference* '(,graph-name ,kb-name)))
```

```
    (kbgoto kb-name)
    (send window :label-graph-window current-graph)
    current-graph)
   (t
    (send window :user-message
          (format
            nil
            "Unit ~a, knowledge base ~a, does not exist or is not a graph unit"
            graph-name kb-name))
    (send window :label-graph-window nil)
    nil)))

;***************************************************************************
;    create-graph (window graph)
;
;    Creates a new graph that is not a sub-graph.
;
;        Asks user for name of a knew graph and sends an INSTANTIATE message to
;        to the class definition of a graph-- '(GRAPH GDA-SYSTEM).
;
;        Returns the new graph's unit.reference.
;
(defun create-graph (window current-graph)
  (declare (special graph-name kb-name))
  (cond
    ((null current-graph)
     (setq graph-name nil)
     (setq kb-name nil))
    ((unitreference* current-graph)
     (setq kb-name    (kb.name (unit.kb current-graph)))))
  (setq graph-name nil)
  (tv:choose-variable-values
    '((graph-name "Graph         ")
      (kb-name    "Knowledge base"))
    :label "Create a new graph       ")
  (cond
    ((kbreference* kb-name)
     (kbgoto kb-name)
     (setq current-graph
           (unitmsg '(graph gda-system) 'instantiate))
     (unitmsg current-graph 'name-element graph-name)
     (send window :label-graph-window current-graph)
     (update-graph-window window current-graph)
     current-graph)
    (t
     (send window :user-message
           (format
             nil
             "Unit ~a, knowledge base ~a, does not exist or is not a graph unit"
             graph-name kb-name))
     (send window :label-graph-window nil)
```

```
    nil)))

;;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;   create-sub-graph  (window parentgraph)
;
;   Creates a new graph which is a sub-graph of the graph which is
;   currently being edited.  The new graph is given a default name.
;
;       The user is shown an outline of the graph node and asked to position
;       the graph somewhere within the graph-window.
;
;       The graph is created by sending an ADD-SUB-ELEMENT message to
;       the class definition of graph--'(GRAPH GDA-SYSTEM).
;
;
(defun create-sub-graph (window parentgraph)
  (if
    (or (null parentgraph)
        (null (unitreference* parentgraph)))
    (send window :user-message
         "Cannot create child graph until a parent graph is assigned to window")
    (let*
     ((subgraph (unitmsg parentgraph 'add-sub-element '(graph gda-system)))
      (height   (unitmsg subgraph 'calc-display-height window))
      (width    (unitmsg subgraph 'calc-display-width window))
      (location (send window :get-location width height)))
     (cond
       (location
        (put.value subgraph 'position location)
        (update-graph-window window parentgraph))
       (t
        (unitmsg parentgraph 'remove-sub-element subgraph)))
     )))

;;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;   delete-element (window graph element)
;
;   Deletes an element.
;
;       "Element" is the object the user is pointing to when he selects
;       this function.
;
;       A pop-up menu appears and asks for a confirmation of the delete
;       function.
;
;       An element is deleted by sending a DELETE message to the element.
;
;
(defun delete-element (window graph element)
  (if
    (get-confirmation (format nil "Delete ~a" (unit.name element)))
```

```
    (progn
      (unitmsg element 'delete)
      (update-graph-window window graph))))

;*****************************************************
;   delete-out-arc (window graph out-junction)
;
;   Deletes an output arc.
;
;       There may be multiple arcs that eminate from a single
;       junction, therefore a SELECT-ARC-FROM-SLOT message is sent
;       to the out-junction.  This displays a menu of all of the
;       output arcs and allows the user to select which one he wants to delete.
;
(defun delete-out-arc (window graph out-junction)
  (let*
    ((arc (unitmsg out-junction 'select-arc-from-slot 'out-arc "Select arc to delete"))
     (to  (get.value arc 'to-junction)))
    (if
      (get-confirmation (format nil "Delete arc to ~a" (unit.name to)))
      (progn
        (unitmsg arc 'delete)
        (update-graph-window window graph)))))

;*****************************************************
;   instantiate-junction (window graph)
;
;     Creates a new instance of a junction.
;
;           A pop-up menu asks the user to select either an input or
;           output junction.
;
;           The user is ask to position the junction in the graph-window.
;
;           The new junctions is created by sending a ADD-SUB-ELEMENT message
;           to the graph.
;
(defun instantiate-junction (window graph)
  (if
    (check-current-graph window graph)
    (let*
      ((avail-classes '((input-junction  (in-connector  gda-system))
                        (output-junction (out-connector gda-system))))
       (class (select-from-alist avail-classes "Input or ouput?")))
      (if
        class
        (let*
          ((junction (unitmsg graph 'add-sub-element (unitreference class)))
           (location (send window :get-location 20 20)))
          (cond
            (location
```

```
            (out.value junction 'position location)
            (update-graph-window window graph))
           (t
            (unitmsg graph 'remove-sub-element junction))))))))

;^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
;    instantiate-primitive (window graph)
;
;    Creates an new instance of a primitive.
;
;      Display's a menu of the primitive library members for this type
;      of graph.  If the user selects a member, an instance of that
;      class of primitives is created.
;
(defun instantiate-primitive (window graph)
  (if
    (check-current-graph window graph)
    (let*
      ((grammar (get.value graph 'grammar))
       (library (get.value grammar 'primitive-library))
       (newclass (unitmsg library 'select-from-menu)))
      (if
       newclass
       (let*
         ((newunit (unitmsg graph 'add-sub-element newclass))

          (location (send window :get-location 30 40)))
         (cond
           (location
            (put.value newunit 'position location)
            (update-graph-window window graph))
           (t
            (unitmsg graph 'remove-sub-element newunit)))))))

;^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
;    modify-out-arc-path (window from-junction)
;
;    Allows the user the change the path of an arc. (The path is the visual
;    representation of the arc. an arc's default path is just a straign line).
;
(defun modify-out-arc-path (window from-junction)
  (let
    ((out-arc (unitmsg from-junction 'select-arc-from-slot 'out-arc "Select arc to modify" )))
    (if out-arc
        (let*
          ((to-junction (get.value out-arc 'to-junction))
           (p1 (get.value from-junction 'output-position))
           (p2 (get.value to-junction 'input-position))
           (old-path (get.values out-arc 'path))
           (new-path (send window :update-path p1 old-path p2)))
          (put.values out-arc 'path new-path)
```

```
        (send window :user-message
             (format nil "Modify arc, old path: ~s,  new path: ~s" old-path new-path))))))

;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
;    rename-element (window element graph)
;
;    Allows a user to rename an element.  All new units are given a default
;    name when they are created. this function allows the unit to give
;    more meaningful names.
;
;         Displays a file in pop-up menu that allows the user to enter
;         the new name.
;
;         The name of the element is changed by sending a NAME-ELEMENT message
;         to the element. This renames the unit and all of the sub-elements of
;         the unit.
;
(defun rename-element (window element graph)
  (declare (special name))
  (setq name (get.value element 'firstname))
  (tv:choose-variable-values
    '((name "Element name"))
    :label "Rename element        ")
  (if
    name
    (progn
      (unitmsg element 'name-element name)
      (update-graph-window window graph)
      (send window :user-message (format nil "New name ~a" name)))
    (send window :user-message "No new name")))



;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
;    update-graph-window (window graph)
;
;    Redisplays the graph-window
;
(defun update-graph-window (window graph)
    (send window :clear-window)
    (send window :show-menu)
    (send window :label-graph-window graph)
    (if
      graph
      (unitmsg graph 'draw-sub-elements window)))

;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
;    zoom-in (window newgraph-object)
;
;    Returns the newgraph-object to the graph-window. This allows the user
;    to edit sub-graphs.
```

```
;
;
(defun zoom-in (window newgraph-object)
  (update-graph-window window newgraph-object)
  newgraph-object)

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;    zoom-out   window oldgraph
;
;    Returns the parent graph (SUPER-ELEMENT).  The super-element
;    becomes the new graph assigned to the graph-window.
;
(defun zoom-out (window oldgraph)
  (if
    (check-current-graph window oldgraph)
    (let
      ((parent (get.value oldgraph 'super-element)))
      (cond
        (parent
         (update-graph-window window parent)
         parent)
        (t
         (send window :user-message
               (format nil "Can't zoom-out. ~a is at the top"
                       (get.value oldgraph 'first-name)))
         oldgraph)))))
```

Bibliography

Ackerman, William B. "Data Flow Languages," Computer 15(2): 15-25 (February 1982).

Allworth, S. T. Introduction to Real-time Software Design. New York: Springer-Verlag, 1981.

Barstow, David R. "Domain-Specific Automatic Programming," IEEE Transactions on Software Engineering SE-11(11): 1321-1336.

Brady, Michael. "Artificial Intelligence and Robotics," Artificial Intelligence 26(1): 79-121 (April 1985).

Boehm, B. W. "Software Engineering," IEEE Transactions on Computers C-25(12): 1226-1241 (January 1977). Reprinted in Classics in Software Engineering, edited by Edward Nash Yourdon. 323-361. New York: Yourdon Press, 1979.

Buchmann, Alejandro P. and Tosiyasu L. Kunii. "Evolutionary Drawing Formalization in an Enggineering Database Environment," COMPSAC 79, The IEEE Computer Society's Third International Computer Software and Applications Conference. 732-737. IEEE Computer Society, Long Beach, California, 1979.

Davis, A. L., "An Introduction to Data-Driven Programming Methodology for PS 300 Users," Graphics Programming. Volume 2b of the PS 300 Documentation Set, Salt Lake City, Utah, Evans and Sutherland Computer Corporation, 1984.

Davis, Alan L. and Robert M. Keller, "Data Flow Program Graphs," Computer 15(2): 26-41 (February 1982).

DeMarco, T., "Structured Analysis and System Specification," GUIDE 47 Proceedings, 1978. Reprinted in Classics in Software Engineering, edited by Edward Nash Yourdon. 411-424. New York: Yourdon Press, 1979.

Ehrig, Hartmut. "Introduction to the Algerbraic Theory of Graph Grammars (A Survey)," Lecture Notes in Computer Science 73, Graph-Grammars and Their Application to Computer Science and Biology. 1-70. Springer-Verlag, New York, 1979.

Evans and Sutherland Computer Corporation, "PS 300 Systems Overview," General Information. Volume 1 of the PS 300 Documentation Set, Salt Lake City, Utah, Evans and Sutherland Computer Corporation, 1984.

Evans and Sutherland Computer Corporation, "Function Networks I," Graphics Programming. Volume 2a of the PS 300 Documentation Set, Salt Lake City, Utah, Evans and Sutherland Computer Corporation, 1984.

Fikes, Richard and Tom Kehler "The Role of Frame-based Representation in Reasoning," Communications of the ACM 28(9): 904-920 (September 1985).

Floyd, Christiane. "A Systematic Look at Prototyping", Approaches to Prototyping, Proceedings of the Working Conference on Prototyping, Namur October 1983. 1-18. Springer-Verlag, Berlin, 1984.

Foley, James D. and Andries Van Dam. Fundamentals of Interactive Computer Graphics. Reading MA: Addison-Wesley Publishing Company, 1982.

Intellicorp, Software Development System Active Images User Manual, KEE Version 2.0. Intellicorp, 1985.

Intellicorp, Software Development System Rulesystem2 Reference Manual, KEE Version 2.0. Intellicorp, 1985.

Intellicorp, Software Development System Reference Manual, KEE Version 2.0. Intellicorp, 1985.

Intellicorp, Software Development System User's Manual, KEE Version 2.0. Intellicorp, 1985.

Jacob, Robert J. K. "A State Transistion Diagram Language for Visual Programming," Computer 18(8): 51-59 (August 1985).

Kant, Elaine "Understanding and Automating Algorithm Design," IEEE Transactions on Software Engineering SE-11(11): 1361-1374. (November 1985).

Kunii, Tosyasu L. and Minoru Harada. "SID: A System for Interactive Design," AFIPS Conference Proceedings, 1980 National Computer Conference. 33-40. AFIPS Press, Arlington, Virginia, 1980.

Kunz, John C. and others. "Applications Development Using a Hybrid AI Development System," The AI Magazine: (Fall 1984).

Mostow, Jack. "Toward Better Models of the Design Process," The AI Magazine 6(1): 44-57 (Spring 1985).

Nagl, Manfred. "A Tutorial and Bibliographical Survey on Graph Grammars," Lecture Notes in Computer Science 73, Graph-Grammars and Their Application to Computer Science and Biology. 70-126. Springer-Verlag, New York, 1979.

Partsch, H. and R. Steinbruggen. "Program Transformation Systems," Computing Surveys, 15(3): 199-236 (September 1983).

Pinson, E. N. "Simulation Environment for Robot Software Development," presentation at SPIE's Cambridge Symposium on Optical and Electro-Optical Engineering, Intelligent Robots and Computer Vision, September 19, 1985.

Rich, Charles. "A Formal Representation For Plans in the Programmer's Apprentice," Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Volume 2. 1044-1052. IJCAI, Vancouver, B. C., Canada, August 1981.

Rich, Charles and Howard E. Shrobe. "Initial Report on a LISP Programmer's Apprentice," IEEE Transactions on Software Engineering SE-4(6): 456-467 (November 1978). Reprinted in Interactive Programming Environments, edited by David R. Barstow and others. 443-463, New York: McGraw-Hill Inc., 1984.

Sheil, B. A. "Power Tools for Programmers," Datamation Magazine. 1983. Reprinted in Interactive Programming Environments, edited by David R. Barstow and others. 19-30, New York: McGraw-Hill Inc., 1984.

Symbolics, Volume 1, User's Guide to Symbolic Computers. Cambridge MA. Symbolics, 1985.

Symbolics, Volume 2, Reference Guide to Symbolics-Lisp. Cambridge MA. Symbolics, 1985.

Symbolics, Volume 4, Program Development Utilites. Cambridge MA. Symbolics, 1985.

Symbolics, Volume 7, Programming the User Interface. Cambridge MA. Symbolics, 1985.

Thomson, C. C. "Robot modelling--the tools needed for optimal design and utilization," Computer-Aided Design 16(5): 335-337 (September 1984).

Veen, Arthur H. "Reconciling Data Flow Machines and Conventional Languages," Lecture Notes in Computer Science 111, CONPAR 81 Conference on Analysing Problem Classes and Programming for Parallel Computing. 126-140. Springer-Verlag, New York, 1981.

Waters, Richard C. "The Programmer's Apprentice: A Session with KBEmacs," IEEE Transactions on Software Engineering SE-11(11): 1296-1320. (November 1985).

Waters, Richard C. "The Programmer's Apprentice: Knowledge Based Program Editing," IEEE Transactions on Software Engineering SE-8(1): (January 1982). Reprinted in Interactive Programming Environments, edited by David R. Barstow and others. 464-486, New York: McGraw-Hill Inc., 1984.

Yamaguchi, Kazunori and others. "A Data Flow Language for Controlling Multiple Interactive Devices," IEEE Computer Graphics and Applications, 5(3): 48-60 (March 1985).

VITA

Captain Alan J. Black was born November 19, 1949 in Salt Lake City, Utah. He graduated from high school in 1968 and attended the University of Utah from which e received the Bachelor of Science in Computer Science in 1974. He enlisted in the Air Force in 1977, and received a commission upon graduation from Officers Training School in 1978. Next he was stationed at Keesler AFB, Mississippi. As an technical instructor on a mobile training team, he taught assembly language programming at World Wide Military Command and Control System (WWMCCS) sites in Europe and Hawaii. In 1981 he was assigned to the Armed Forces Radiobiology Research Institute (AFRRI) in Bethesda Maryland. He developed computer programs for scientific investigators doing biomedical research at AFRRI until entering the School of Engineering, Air Force Institute of Technology, in June 1984. He is a member of Tau Beta Pi.

SECURITY CLASSIFICATION OF THIS PAGE

ADA 164122

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; Distribution unlimited;
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GCS/ENG/85D-2	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	
12. PERSONAL AUTHOR(S): Alan J. Black, B.S., Capt, USAF	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM _____ TO _____; 14. DATE OF REPORT (Yr., Mo., Day): 1985 December; 15. PAGE COUNT: 173	
16. SUPPLEMENTARY NOTATION	

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB GR	
09	02		Automatic Programming, Dataflow, Program Transformation System, Robot Simulation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: A PROGRAMMER'S ASSISTANT FOR A SPECIAL-PURPOSE DATAFLOW LANGUAGE

Thesis Chairman: Gary B. Lamont
Professor of Electrical Engineering

Approved for public release: IAW AFR 190-1.
LYNN E. WOLAVER 10 JAN 86
Dean for Research and Professional Development
Air Force Institute of Technology (AFIT)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED [X] SAME AS RPT. [ ] DTIC USERS [ ]	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Gary B. Lamont	22b. TELEPHONE NUMBER (Include Area Code): 513-255-3576 — 22c. OFFICE SYMBOL: AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

Abstract

A programming tool, the Graph Design Assistant (GDA), for a special-purpose dataflow language was designed and implemented. The motivation for the effort was the need to construct a robot simulation facility which will assist in the development of effective algorithms to plan and control robot movements.

An Evans and Sutherland PS300 graphic workstation is used to display animated robot simulations. The Graph Design Assistant was developed so that researchers could program robot simulations on the PS300 without having to learn the intricacies of the PS300's dataflow language.

The Graph Design Assistant was implemented on a "Lisp machine" using a knowledge engineering tool. The result of the effort was a prototype system to be used as the basis for further development.

END

FILMED

3 -86

DTIC