Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

GRAPHICAL DEEP KNOWLEDGE REPRESENTATION IN VODAK/VML OBJECT-ORIENTED DATABASE

By

Jue Wang

Applying Object-oriented concepts to the design of complex graphical interface has received great attention in the database and knowledge representation disciplines. Traditional CAD systems can not support efficient environments for design processes because they store information about all the objects for display purposes but do not store any knowledge for reasoning purposes. They are called "knowledge poor". "Graphical Deep Knowledge" in Artificial Intelligence has been proven successful to represent knowledge about objects for display purposes as well as reasoning purposes. We introduced the theory of "Graphical Deep Knowledge" into the object-oriented database system VML to design a "Knowledge rich" system which can support better graphical interface. We showed that the theory of "Graphical Deep Knowledge" can improve graphical interface design and functions as a flexible, and knowledgeable tool for design processes. Based on the presented theory, we developed a system called GDKRIVVOOD (Graphical Deep Knowledge Representation In Vodak/VML Object-Oriented Database) for circuit board design.

GRAPHICAL DEEP KNOWLEDGE REPRESENTATION IN VODAK/VML OBJECT-ORIENTED DATABASE

By

Jue Wang

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

May 1993

APPROVAL PAGE

Graphical Deep Knowledge Representation in VODAK/VML Object-Oriented Database

Jue Wang

5.13.93

Dr. Yehoshua Perl, Thesis Advisor (date)
Professor, Department of Computer and Information Science,
New Jersey Institute of Technology

5/14/1993

Dr. Jason T. L. Wang, Committee Member (date)
Assistant Professor, Department of Computer and Information Science,
New Jersey Institute of Technology

5.13.93

Dr. Daniel Yuh Chao, Committee Member (date)
Assistant Professor, Department of Computer and Information Science,
New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author: Jue Wang

Degree: Master of Science in Computer Science

Date: May 1993

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Master of Science in Computer Science, New Jersey Institute of Technology, Newark, New Jersey, 1993
- Bachelor of Science in Computer Science, Zhejiang University, Hangzhou, Zhejiang, China, 1984

Major: Computer Science

This work is dedicated to my parents.

ACKNOWLEDGMENT

The author wishes to express her sincere gratitude and thanks to her advisor, Dr. Yehoshua Perl, for his guidance, friendship, and moral support throughout this work. He accepted me into his research group and provided me with the opportunity to work in a relatively new area. My acknowledgements are also due to all the participants of Dr. Perl's research group meetings who made some very valuable suggestions, without which this thesis would not be a success.

Special thanks to Mr. Ashish Mehta who guided us in the design and implementation of the whole system.

Another special thanks goes to my team member Aruna Kolla for her friendly cooperation and suggestion.

Sincere thanks to Dr. James Geller and Michael Halper for their advisement.

Sincere thanks to Dr. Jason Wang and Dr. Daniel Yuh Chao for their encouragement and suggestion.

And finally, a thank you to Bill Wu and Xiaowen Mang for their help.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Object-oriented Databases that support design automaton have received a great attention and become increasingly popular. A key component of such design databases is designing a graphical interface. Usually, There are two major limitations of such design environment as follows:

1. Traditional database systems do not support retrieval and manipulation of complex objects, which results in an inefficient graphical interface.

2. Traditional CAD systems store information about objects for display purposes but do not store any knowledge for reasoning purposes. Such interfaces are called "knowledge poor"

Many authors[8][9][10] proposed to solve the first limitation using object-oriented databases. One of the major advantages of object-oriented databases is that they provide the mechanisms of retrieval and manipulation of complex objects. Traditional CAD applications only store information about objects for display purposes, i.e., lines, points, etc., but they do not store any knowledge about objects for reasoning purposes. For example, if we want to know how many gates there are in a circuit, the traditional system might not be capable of answering such a query.

Similar concerns[11] led to the introduction of the notion of *Graphical Deep Knowledge* in Artificial Intelligence[3][5][4]. This notion can represent knowledge about objects for display purposes as well as reasoning purposes. It was described in terms of the SNePS (= Semantic Network Processing system) semantic network representation [12]. The SNePS system can be viewed as a relational system, but Graphical

Deep Knowledge is system independent and can be easily applied to object-oriented systems as well.

In our system GDKRIVVOOD(Graphical Deep Knowledge Representation In Vodak/VML Object-Oriented Database), we represent Graphical Deep Knowledge in an object-oriented database system which is used for a graphical interface. In this way we can overcome the limitations of traditional CAD systems and achieve a flexible, knowledgeable graphical interface. We look for a "Knowledge rich" system and maintain Graphical Deep Knowledge in the VML (= VODAK Data Modeling language, [2]) object-oriented database system.

The remainder of the chapters are organized as follows: The background of the system is reviewed in chapter 2, which focuses the Dual model, Graphical Deep knowledge and previous work. In chapter 3 the database system which was developed in VODAK/VML will be described in detail. In chapter 4 the interface of the system which was developed in C++ and Xwindow will be presented in a top-down way. Finally, a brief conclusion will be given in chapter 5.

CHAPTER 2

THE BACKGROUND OF THE SYSTEM

The theoretical background of this system which has already been mentioned in the previous chapter includes the "Object-oriented Dual model" and "Graphical Deep Knowledge". Also, the current prototype which is implemented in a C++/database/ object-oriented/dual model environment is based on previous VML prototype which was based on Smalltalk.

2.1 The Dual Model

The Dual Model[1] is an object-oriented database model in which the object type description is separated from the description of the object class, thereby enhancing the distinction between the structural and semantic elements in the definition of an object class. Dr. Perl and Dr. Geller of NJIT[1] and Dr. Neuhold of GMD-IPSI proposed this theory. They stated that traditionally two classes that are semanticly similar must also look structurally similar. They also argued that this condition seems practically too limiting and theoretically not justified. They achieved a better abstraction mechanism by separating structural(object types) and semantic(classes) aspects of OODBs.

The specification of object types and classes is made up of four properties: (1) attributes, (2) relationships, (3) methods, and (4) generic relations. The difference between (2) and (4) is that generic relations are predefined and known to the system, while relationships are user defined.

Generic relationship can be divided to semantic relationship and structure relationship as following:

1. Semantic Relationship

   - *roleof*: It is used to express that two object classes model the same real world object in different contexts(or equivalently the objects are represented in different roles).
   - *categoryof*: It is used to model the same real world object with additional knowledge, but still in the same context.

2. Structural Relationship

   - *memberof* & *setof*: If one object type is a set type that consists of elements of another object type, then the connection between the set type and the element type can be expressed by *setof* and *memberof*.

Figure 2.1 is a typical example of the relationships in our OODB:

Semantic relationships are depicted with links. In figure 2.1, the right top part illustrates the *categoryof* relationship which is represented by a solid arrow, in this case, *digital_component* is a category of *component*. The left bottom part of the figure illustrates the *roleof* relationship which is represented by a dotted arrow, in this case, *board_component* is a role of *component*. The right bottom part illustrates the ordinary relationship *belongsto* which is represented by a solid arrow labeled *belongs_to*, in this case, component belongs to a pc_board.

Structural relationship is depicted in the figure with their geometric features and positions. A rectangular, for example, *component*, represents an object type and a double rim rectangular, for example, *components*, represents a set type of its neighboring rim object type, in this case, the *component* object type. These two rectangulars share a corner.

Now, we will explain a few class definitions in the Dual Model.

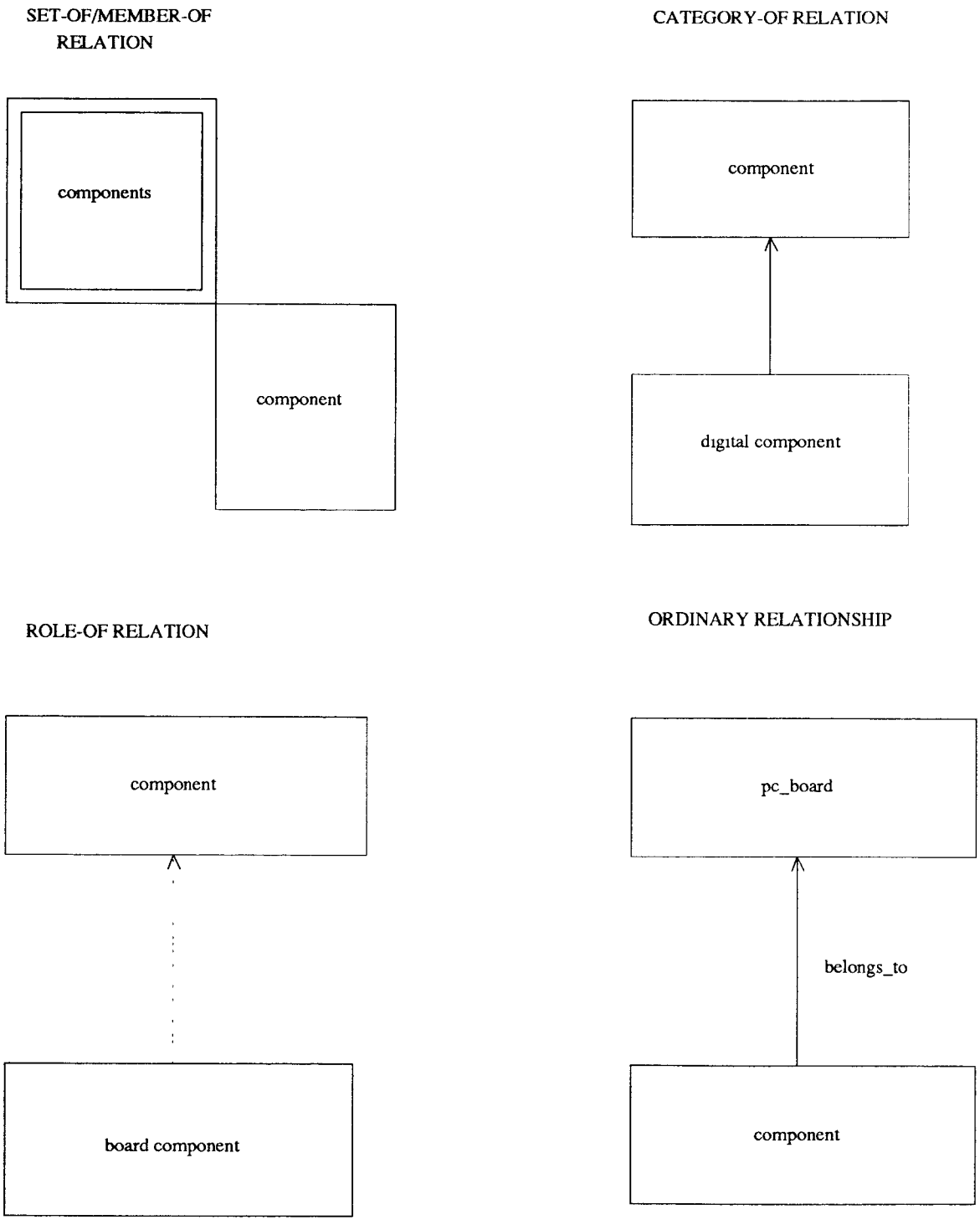

Figure 2.1: Semantic and Structural Relationships

objecttype COMPONENT
- **memberof:** COMPONENTS
- **attributes:**
  - Name: STRING
  - Modality: STRING
  - Color: INTEGER
  - Label: STRING
  - Size: INTEGER
- **relationships:**
  - Pins: PINS

class component
- **objecttype** COMPONENT
- **essential:** Label

objecttype DIGITAL_COMPONENT
- **subtypeof:** COMPONENT
- **memberof:** DIGITAL_COMPONENTS
- **attributes:**
  - Outputs: INTEGER
  - Inputs: INTEGER

class digital_component
- **objecttype** DIGITAL_COMPONENT
- **categoryof:** component

objecttype BOARD_COMPONENT
- **memberof:** BOARD_COMPONENTS
- **attributes:**
  - Position: POINTTYPE
  - Area. RECTANGLETYPE
- **relationships:**
  - Component_of: PC_BOARD
  - Component_connected_to. CONNECTIONS

class board_component
- **objecttype** BOARD_COMPONENT
- **roleof:** component

objecttype COMPONENTS
- **setof:** COMPONENT
- **attributes:**
  - Cardinality: INTEGER
  - Purpose: STRING

class components
- **objecttype** COMPONENTS

Figure 2.2: An Example of a Schema

Figure 2.2 illustrates the use of object type and object class definitions, the structural subtype hierarchy and the semantic *roleof* and *categoryof* relations.

The names of the object classes are written in small letters, the names of the object types are printed with capital letters, names for attributes and relationships have only the first letter capitalized and keywords are in bold face. Some composite data types are used. Their exact definitions are omitted. Their names are self-explanatory. The two columns describe object types (left) and classes (right).

The COMPONENT object type is the supertype of the object type DIGITAL_-COMPONENT. Thus, object type DIGITAL_COMPONENT inherits the attributes Name, Modality, Color, Label and Size. It also inherits the relationship Pins. The COMPONENT object type is also a member type of COMPONENTS.

By using the structural relationship "setof" we obtain the set of components. The attribute Cardinality is a counter for the number of components of a board. The attribute Purpose states the purpose of the set of components. Note also that the DIGITAL_COMPONENT object type is a member type of DIGITAL_-COMPONENTS.

Some component may be on a board and some may not. To capture this semantic a general object class component is introduced. The class board_component will be defined as roleof the class component.

Some components may be a special category of components, digital or analog. To capture this semantic, for example, we define a class digital_component as a category of class component.

The essential Label specifies that each instance of the class component has non-nil value of Label.

2.2 Graphical Deep Knowledge

Graphical Deep Knowledge can be defined [3][5][4] as follows: A Knowledge base is said to contain Graphical Deep Knowledge if at least part of its knowledge exhibits *deductive graphical adequacy* (a knowledge base which can be done propositional, i.e. "logical", reasoning), and part of its knowledge exhibits *projective adequacy* (propositional knowledge that can be used to create diagrams) [3].

The main point of the definition of Graphical Deep knowledge is that the same knowledge is used to represent a designed object internally for both reasoning purposes and display purposes. Therefore, the graphical representation can never be "out of synch" with the rest of the internal model. The notion of Graphical Deep Knowledge also implies that the diagrammatic representation of an object is created mostly as a "byproduct" of the modeling activity.

The representational requirements of Graphical Deep Knowledge can be summarized as follows:

- The representation should be projectively adequate.
- The representation should be deductively adequate.
- The representation should be based on conceptual primitives which seem natural to the human observer.
- The representation should support relation between primitives in case the relation is natural to humans.
- The representation may contain redundant but consistent information.

The heart of the Graphical Deep Knowledge is the use of "form object" which function as an argument in the propositional representation and at the same time as hooks into the graphics environment. Every "form object" has a corresponding icon

which can be described by its name. For instance, a form object called ANDGATE should be connected to a drawing routine that will draw the symbol of an AND gate.

2.3 Previous Implementation

The previous version of the implementation is named "AKROOTIS" by Prasanna S Venkatesh in 1991. "AKROOTIS", which stands for "A Knowledge Representing Object-Oriented Tool In Smalltalk", is a kit to design, edit, display and query boolean circuit boards. AKROOTIS was implemented using the previous version of VML, on a Sun 3/140 workstation. The old version of VML was implemented on top of Smalltalk. Therefore, the previous implementation of the interface used Smalltalk browsers and the Smalltalk method syntax, and the Smalltalk graphics environment. AKROOTIS is modeled along the lines of "TINA" the original Graphical Deep Knowledge interpreter[6].

CHAPTER 3

DEVELOPING AN OBJECT-ORIENTED DATABASE USING THE NEW VML PROTOTYPE

First of all, let's discuss several issues for database schema design for logical circuits. We need to come up with a class hierarchy for our domain. Each class has to be assigned proper attributes and relationships such that our system can maintain knowledge about the circuit it represents and how to display it. The system should also maintain knowledge about the types of components, their parts, their connections and their attributes. In addition to this, the system should maintain the knowledge about the graphical display of the circuit as well as for each component. While those tasks are difficult in traditional database this is exactly the problem for which Graphical Deep Knowledge has been introduced as a solution.

3.1 VODAK Database System and VML Language

The database system which is mentioned above is developed under VODAK and VML[2]. VODAK is a distributed object-oriented database system which is under development at GMD-IPSI. VML is an object-oriented database model language developed for VODAK, integrating the components DDL, DML, and the programming language. The previous prototype of VML was written in Smalltalk. The current prototype available to us was written in C++.

One of the features of current prototype of VML is the support of the Dual Model, i.e. the separation of structural and semantic aspects in the class definition. This feature allows us to achieve a better abstraction when two classes look structurally similar but differ semantically.

VML is also a typed language, i.e., type checking as much as possible. It is an open data model, i.e., providing features to tailor the model to specific application needs. Finally, it provides the concept of open nested transactions, object-oriented serializability(incorporating semantics of methods).

The notation of *metaclasses* were also introduced into the VML. Metaclasses permit us to bridge the gap between generic relations and relationships by defining the semantics of relations in classes which have themselves classes as instances.

3.2 Circuit Board Database Schema

In this section, we discuss the circuit board database schema described in VML. Figure 3.1 is a part of the whole schema using the same notation as described in section 2.1. In addition, the ellipses represent properties.

In VML, a new class may be defined with the following declaration.

```
CLASS component
  INSTTYPE componentType [ component, components, pins ]
END;
```

Here we define the class component. The INSTTYPE clause states that the type of the instance of the class, in this case, component, is componentType. The object type can be defined as follows.

```
OBJECTTYPE componentType [
  componentClass : componentType,
  componentsClass : componentsType,
  pinsClass : pinsType ]
    SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Name : STRING;
    Modality : STRING;
    Color : INT;
    Label : STRING;
```

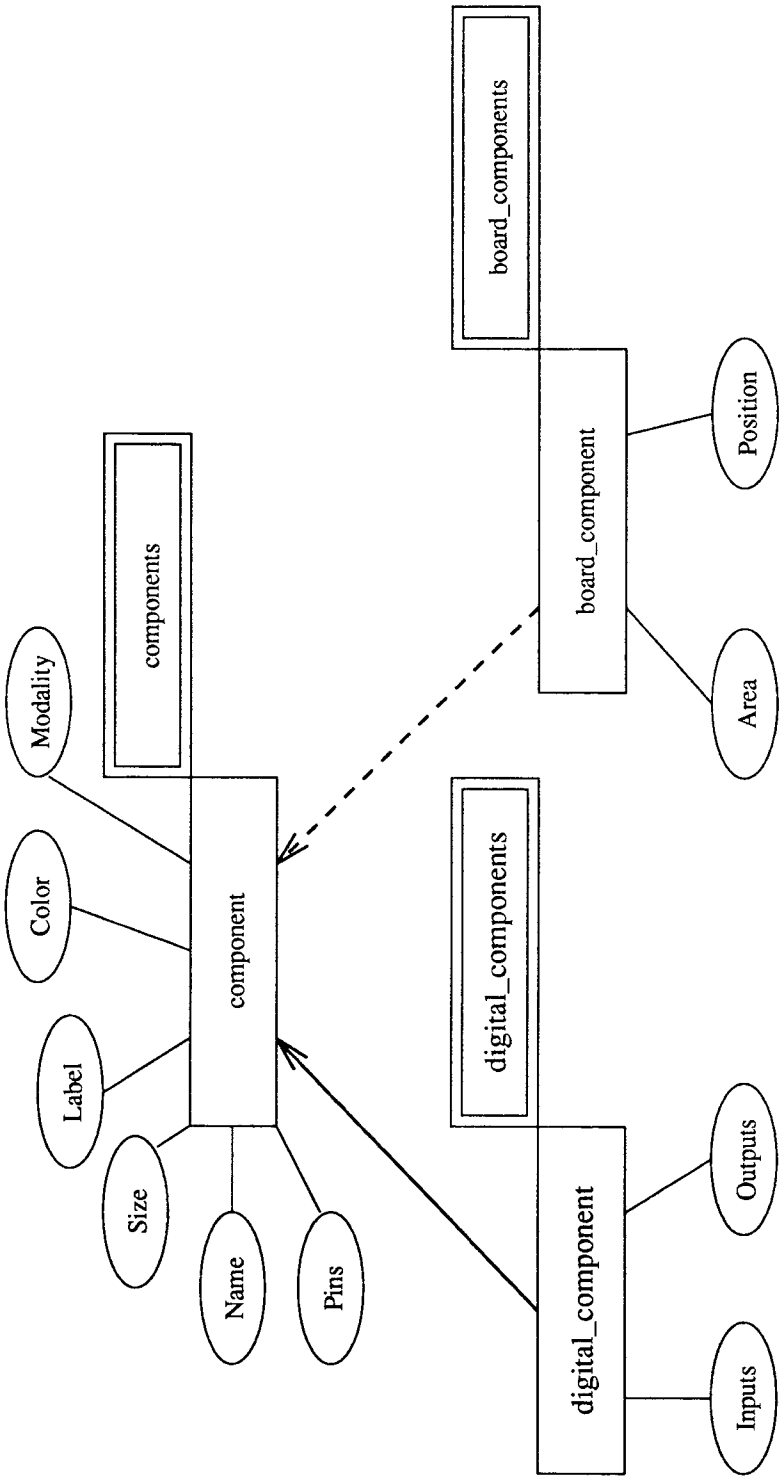

Figure 3.1: A Part of Database Schema

```
    Size : INT;
    Pins : pinsClass;
    memberof : componentsClass;
  IMPLEMENTATION
  METHODS
    method_componentType() READONLY; {};
END;
```

The object type componentType has 7 properties. *Name* is a string referring to the name of the component. *Modality* is a string referring to the modality of the component. *Color* is an integer indicating the color of the component. *Label* is a label assigned to the component. *Size* contains the length of the bounding box of the component. *Pins* is a set of pins that are related to the component. *memberof* is a relationship property. It indicates that a component is a member of object components of type componentsType. Recall that in section 2.1, *memberof* and *setof* are defined directly using object types in the Dual Model. In VML, however, object type names are not referred directly in the relationship declaration. In this example, object type componentType is a member type of componentsType and object type componentsType is a set type of componentType.

In addition to the class component, we have some other classes. Three of them, which are similar to that of component, are illustrated as follows. See appendix for complete definitions.

```
CLASS components
  INSTTYPE componentsType [ components, component ]
END;

OBJECTTYPE componentsType [
  componentsClass : componentsType,
  componentClass : componentType ]
    SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Purpose : STRING;
    Cardinality : INT;
```

```
    setof : componentClass;
  IMPLEMENTATION
  METHODS
    method_componentsType() READONLY; {};
END;
```

Class components is declared to be of type componentsType. The componentsType in turn has 3 properties. *Purpose* is a string referring to the purpose of this set of components. *Cardinality* is an integer containing the number of components in the set. *setof* is a generic relationship. It indicates that an instance of object type components is a set of instances of type componentType. Once again, we see that the declaration of the setof relationship does not refer to the object type name directly.

```
CLASS digital_component METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE digital_componentType [
    digital_component, digital_components,
    component,components,pins ]
  INIT  digital_component->defCategory( component, aspect  )
END;

OBJECTTYPE digital_componentType [
  digital_componentClass : digital_componentType,
  digital_componentsClass : digital_componentsType,
  componentClass : componentType,
  componentsClass : componentsType,
  pinsClass : pinsType ]
    SUBTYPEOF componentType [
      componentClass, componentsClass, pinsClass ];
  INTERFACE
  PROPERTIES
    Outputs : INT;
    Inputs : INT;
    memberof1 : digital_componentsClass;
  IMPLEMENTATION
  METHODS
    method_digital_componentType() READONLY; {};
END;
```

Class digital_component is a category of class component. Its object type is digital_componentType . The *categoryof* relationship is defined using a META-CLASS CATEGORY_SPECIALIZATION_CLASS clause and is implemented using an INIT clause. The object type digital_componentType in turn is a subtype of componentType, which is defined using a SUBTYPEOF clause. Since digital_componentType is a subtype of componentType, it inherits all properties of componentType. In addition, it has 3 more properties. *Outputs* is an integer containing the number of outputs of the digital component. *Inputs* is an integer containing the number of inputs of the digital component. *memberof*1 is a generic relationship. It indicates that a digial component is a member of an instance of type digital_componentsType.

```
CLASS board_component METACLASS ROLE_SPECIALIZATION_CLASS
  INSTTYPE board_componentType [
    board_component, board_components, connections, pc_board ]
  INIT  board_component->defRoleClass(  component  )
END;

OBJECTTYPE board_componentType [
  board_componentClass : board_componentType,
  board_componentsClass : board_componentsType,
  connectionsClass : connectionsType,
  pc_boardClass : pc_boardType ]
    SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Position : POINTTYPE;
    Area : RECTANGLETYPE;
    Component_of : pc_boardClass;
    Component_connected_to : connectionsClass;
    memberof : board_componentsClass;
  IMPLEMENTATION
  METHODS
    method_board_componentType() READONLY; {};
END;
```

Class board_component is a role of class component. Its object type is board_-componentType . The *roleof* relationship is defined using a METACLASS ROLE_-

SPECIALIZATION_CLASS clause and is implemented using an INIT clause. The type board_componentType has 5 properties. *Position* is the location of the the component. *Area* is the bounding box of the component. *Component_of* is a relationship property. It indicates that a board component is a component of a pc_board. *Component_connected_to* is also a relationship property indicating that a board component can be connected to other components through a set of connections of class connectionsClass. The *memberof* is another generic relationship. It indicates that a board component is a member of instances of type board_componentsType.

CHAPTER 4

DEVELOPING USER INTERFACE USING X-WINDOWS AND C++

4.1 The Functions of the Interface

Generally, the function of the interface is to aid users to design, update, query and maintain their circuit board layout. The functions can be categorized as (1) Display Purpose Queries, (2) Reasoning Purpose Queries. Display purpose queries supported in our system are:

- Draw Components: It includes drawing *And* Gate, *Or* Gate, *NAnd* Gate, *Nor* Gate, *Not* Gate, *Xnor* Gate, Buffer, Resister, Power supply, Capacitor, Resistor Log, *PNP* Transistor, *NPN* Transistor, *PNP* Fet, *NPN* Fet, etc. The way to draw a component is firstly selecting the component you want to draw by clicking the mouse on one item in the item list on the left of the screen, then clicking the mouse again in the drawing area to specify the location of the component to be drawn.

- Connect Components: Any pair of components can be connected from one output pin of one component to an input pin of the other. The way to draw the connection line is clicking the mouse on both involved pins. After the first pin is clicked, the connection line appears from the selected pin to the mouse cursor in order to highlight the current drawing activity. After the second clicking on the second pin, either a connection line is automatically drawn or an error messages is shown in the text area.

- Disconnect Components: It is a reverse operation of Connection. This operation will remove a wire connecting two pins. To disconnect two components, click the mouse on both involved pins. After the second clicking on the second pin, either the connection line is removed or if there is no connection between the two clicked pins an error message is shown in the text area.

- Move Components: Any component in the drawing area can be moved from one location to another by clicking the component and the new position. In the meantime, its connection lines are automatically redrawn to meet the new location of the components.

- Delete Components: Any component can be deleted by first clicking "Delete Component" and then clicking the component being deleted. After successful deletion, all connection lines related to the component are automatically deleted.

These kind of queries are available for graphical interface particularly in CAD systems.

In addition to the above display purpose queries, our system supports reasoning purpose queries. It is necessary to know how many gates of a particular type are used in a circuit to select proper Integrated Circuits. This can be achieved by selecting the query function and the NumberOfGates option in the pulling down menu. The query result is shown within the message window. It is also natural to generate a list of components used in our system. This can be achieved by selecting the query function and the ListOfComponents option. Another option in the pulling down menu is the query of component positions.

One more reasoning purpose query is the modality of the circuit. The drawing is logical in the sense it only specifies the logical connections between components. However, users may want to know the physical appearance of the component. Physical appearance can be deducted from the logical relations using some rules. In our system,

we use a simple rules about the number of pins of the component. Simply speaking, the total number of pins of the circuit should be equal to the total number of output pins and input pins plus 2 pins for power supply. The reasoning purpose query are categorized as:

- Physical Modality & Logical Modality[3]: When modality in the left top corner of the screen is clicked, there exist two options. One is for displaying physical modality of the current circuit, the other is for displaying logical modality of the current circuit board.

- Query: This option includes Querying the number of components, listing components and displaying component's positions.

It should be mentioned here that, the drawing area which appears in front of the users is only a window of the actual drawing area, which is much larger than the window appearing on the screen. Users can use the scroll bar to shift the drawing window to an appropriate location of the drawing area when they are designing a large layout. If the drawing area is not large enough, user can ask the designer to extend it. Also the text area which is used to display query information is larger than appear and can be extended.

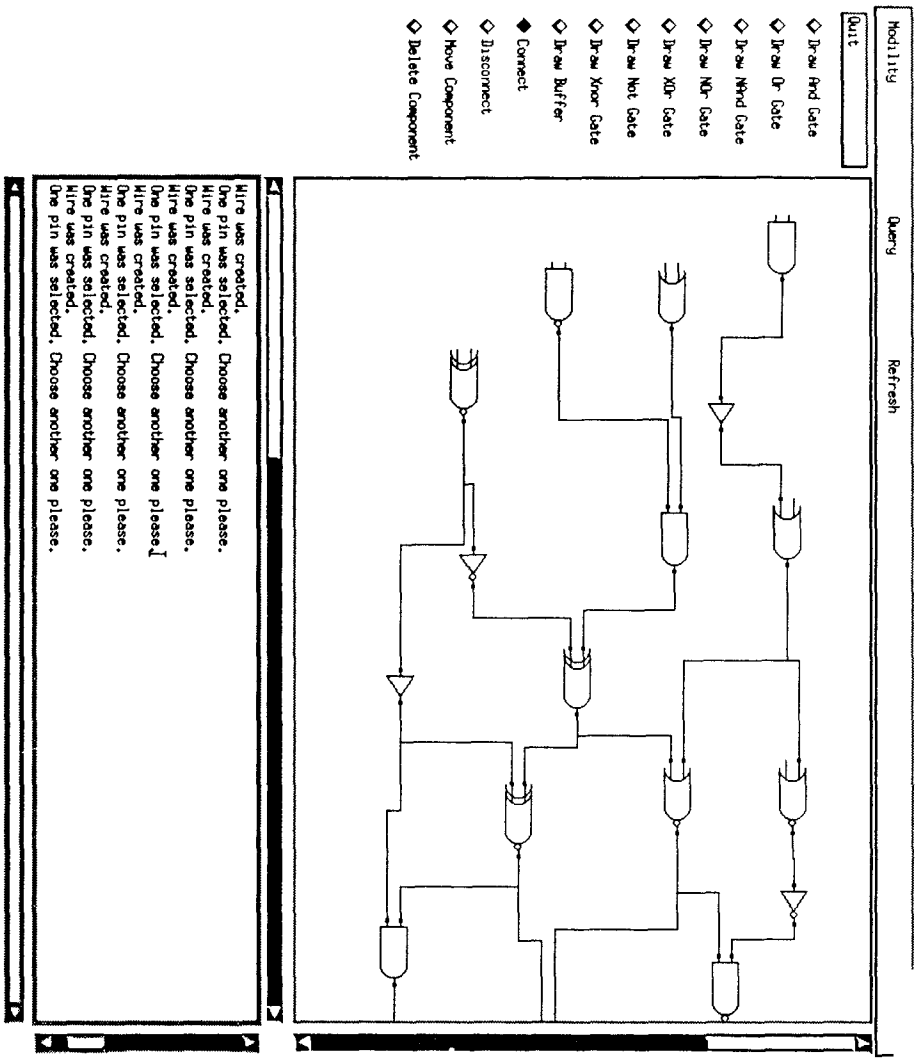

Figure 4.1: Window Interface

4.2 The Design Issues of the System

4.2.1 The Outline of the Program

Basically, this part of the program can be simplified as following:

```
main
{
 Define all windows that will appear in front of a user;
 Define relationship among all windows;
 Initialize and activate the top windows;
 Application loop to wait for user input;
}
```

In Motif window system, windows are identified and represented by so called widget variables, or simply widgets. There are a lot of widget types available and each of them is suitable for different situation. For example, a selection menu is called a *command* widget. For more detailed information, please see the source code and Motif user manual.

The relation among widgets are also very important. In a user interface, there are several kinds of relations between two widgets: relative location relation, parent-children relation, control relation, and so on. These relations are represented in a widget hierarchy and defined using a library of X and Motif functions.

After all windows and their relations are defined, the top window is activated. The top window controls the whole program. One of its task is to transfer user input to the appropriate windows. It should be noted that the top window is introduced only for functional purpose. It is invisible.

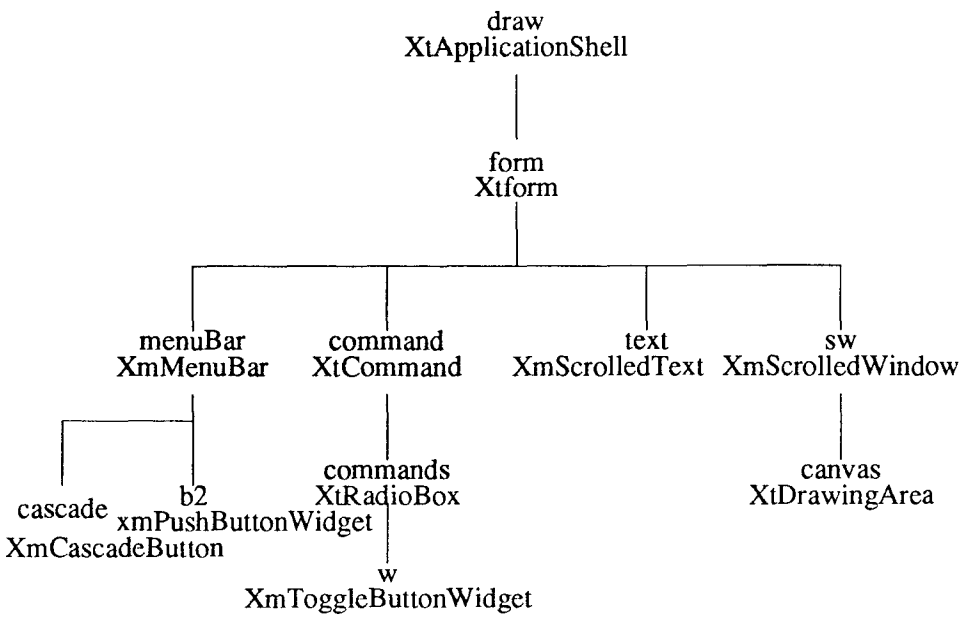

Figure 4.2: Widgets Hierarchy

4.2.2 Widgets Hierarchy

The widget hierarchy represents the relationships among widgets. Figure 4.2 shows the hierarchy tree of the widgets.

The top widget is *XtApplicationShell* which is responsible for controlling the whole window, such as the location and size within the display (geometry as termed in X-window system).

The form widget *framework* is a container which contains action menu, command list, message area, drawing area. It allows the designer to specify the relative relation among the windows it contains. For example, the command list must appear on the left of both the drawing area and the message area. The drawing area must appear immediately above the message area. Such relative location relations are maintained by the *framework* when the size or the location of the involved windows are changed.

The first functional window is the *menuBar* widget. Users can click it to pull down a menu. From the menu, users can issue a number of requests, which are usually not related to the designing of the circuit board layout.

The second functional window is the *command* widget. This widget has two child widgets, one is the *quit* button which is specially designed in the Motif, the other is a *RadioBox* widget which serves as a command list. The *RadioBox* widget is responsible for ensuring that only one command in the list may be active at any time. It is also responsible for arranging all the commands in a column.

Managed by the *RadioBox* widget, each command in the command list is a *ToggleButton* widget implying that it has only two states, active or inactive. Because their parent widget *RadioBox* ensures that only one command in the list may be active at any time, a command will become inactive automaticly whenever another command becomes active upon a user's request.

The command list contains drawing instructions. Each command in the list corresponds to a drawing activity, such as draw an *And* gate, delete a gate, etc. Associated with each command is a small diamond, termed *ToggleButtonWidget* in Motif. This diamond indicates if the command is currently active. In order to perform a desired drawing, a user must at first make the corresponding command active by clicking the widget of that command. When a command is active, users can perform the drawing activities without clicking the command every time when the same drawing activities are being performed continuously. Although this feature saves time, users should be alert enough when clicking in the drawing area, especially when some unrecoverable commands are active. For example, when delete gate command is active, each time the user clicks a gate in the drawing area, the clicked gate and all connection lines related to it will be deleted.

The third functional window is the text message window. It serves as a speaker of the system. Users are urged to pay attention to the message delivered by the system

to inform the user what the system is doing and what the system requires. The text window keeps some amount of old messages. If the user wants to see the old messages that are out of the window, he/she can use the scroll bar to scroll forward/backward and left/right.

The last but the most important part is the drawing area termed *canvas* in our system. It serves as a drawing board in real life. Only part of the drawing area is visible at one time. This part is called *ScrolledWindow* which manages the *canvas* which keeps the whole drawing. Users are able to shift the *ScrolledWindow* to any part of the *canvas*. Current drawing can only be performed on the displayed part of the *canvas*.

4.2.3 Associate Callbacks with Different Widgets

In section 4.2.1, we can see that the system enters loop waiting for user input after all definition are done. There is no sequence of instructions on what the system will run next. This is the style of X-window programming called "event driven programming environment". This means the user interface is programmed to respond to user "events". When the user hits a key on the keyboard, this generates an event that the program must then respond to. When the user moves or clicks the mouse, events are generated. An event loop sits and waits for user events. When events occur, the event loop calls the correct piece of code to handle them. We try our best to utilize Motif since the Motif code handles almost all of the low-level user-generated events, updates the screen based on those events, and then (after all the hard and tricky stuff is done) it calls the code we have written, termed callbacks. By using Motif, we put most of our time in designing data structures, algorithms and the callbacks. Callbacks are the places where the designers write code to process user inputs. They are also very important in the sense that they are the only way to incorporate our

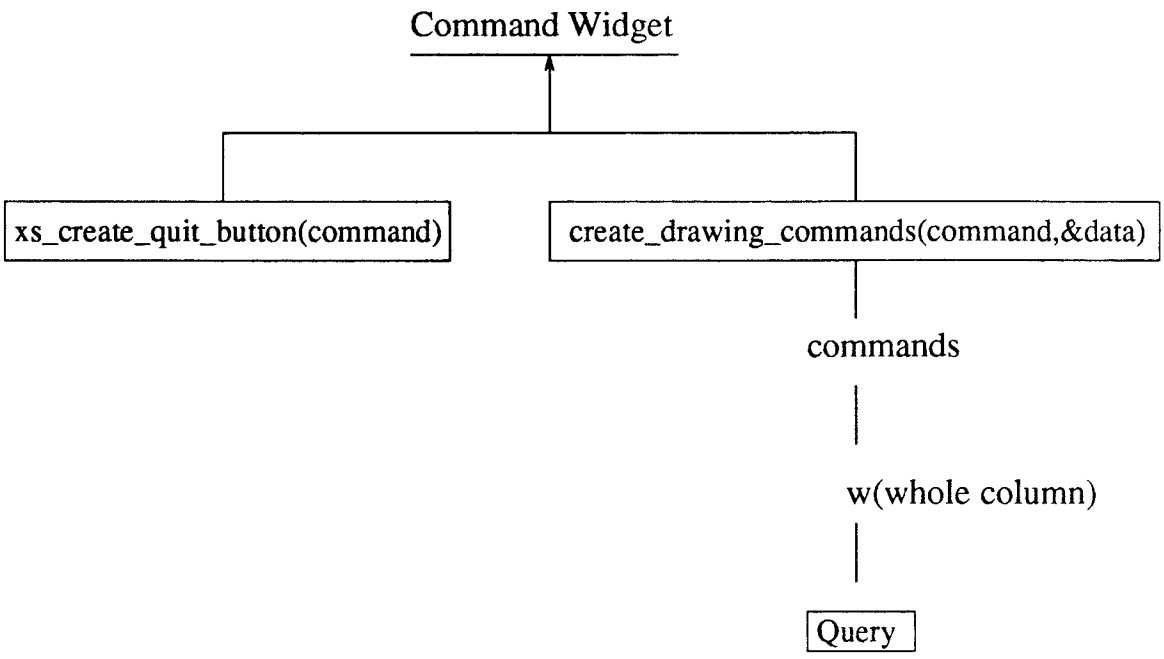

Figure 4.3: Associate Functions and Callbacks with Command Widget

interface system with the backend database systems.

We have to write callbacks for almost all widgets involved. In this section, all of them will be discussed in turn.

Figure 4.3 shows the functions dealing with the command widget. The callbacks of the command widget are built in for Motif. When a user clicks any command in the command list, a Motif defined callback determines which command is clicked and then sets appropriate internal variables. After that, an event is generated and is sent to the widget corresponding to the clicked command so that the callbacks for that widget will be called. We defined the callbacks associated with each drawing command. Though in general each command can have different callbacks, in our system, all commands have the same callback function, *activate*(). When a command widget's callback is called, the first argument is the widget pointer to itself. The function *activate*() simply sets a global variable $current_func$ to its associated function

Associate callbacks and functions with canvas widget				
	start_rubber_band	track_rubber_band	end_rubber_band	refresh
Draw Objects	new AndGate(x,y) gate→drawpic()	gate→move(x,y)	gate→settle() store_object()	
Move Objects	StartMovingGate(x,y)	MoveGate(x,y)	StopMovingGate()	
Connection	wire→drawTo(x,y)	wire→move(x,y)	store_wire()	
Disconnection	wire→drawTo(x,y)	wire→move(x,y)	delete_wire()	
Delete Gates	DeleteGate()	—	—	

Figure 4.4: Associate Callbacks and Functions with Canvas Widget

number so that the whole system will know which command is currently active. A function number is one of the resources of a command widget. They are defined when the widget is created. See $create_drawing_commands()$ in the source codes for detail. It should also be noted that the appearance of the activeness of command widgets is handled by the Motif.

The most important and complicated widget is the *canvas*. The callbacks must handle all mouse clicking. There are three callbacks. They are *start_rubber_band*, *track_rubber_band* and *end_rubber_band*. The *start_rubber_band* is invoked whenever the mouse button is pressed. The *track_rubber_band* is invoked whenever the mouse is moved while the mouse button is held.

The *end_rubber_band* is invoked whenever the mouse button is released. Once the mouse button is pressed, a drawing activity is initiated, such as drawing a gate, connecting two pins, etc. While holding the mouse button, users may move the mouse to choose a location or find the second pin. Once the mouse button is released, the system will try to complete the drawing activity. Sometimes, the system will not be able to complete the drawing because the user's action does not comply with the information the system has. This may happen when a user fails to click close enough

to the pin he/she wants to select. When the drawing can not be completed, the system will abort the activity and inform the user.

Before we discuss the three callbacks in detail, we must first discuss how a drawing is performed on the canvas. Up to now, our drawing area is a black/white area. Each pixel has only two possible states, white or black. There are three operations on a pixel, set to the foreground, set to the background and xor the pixel with a given value. The first two are used only when a permanent drawing is desired because they will remove any old information on the screen. Xor the pixel with a given value is used when temporary drawing is desired. When the Xor is completed, the old pixel is xor'ed with the given value. The resulting pixel is xor'ed again with the given value when the temporary drawing is to be removed. Since $x \oplus x = 0$, let p is the old value of the pixel, q is the temporary drawing value, then after the first xor, the pixel has the value $p \oplus q$, after the second the xor, the pixel has the value $(p \oplus q) \oplus q = p$. So the old pixel is restored.

When a drawing activity is initiated, the xor operation is used. After the activity is completed, the set to foreground operation is used. When the deleting activity is being performed, the set to background operation is used.

Each of the three callbacks works in a similar way. They all at first check which command is active and then perform the corresponding tasks.

In *start_rubber_band*, there are several operations according to the current active command. If the active command is drawing a gate, a new object representing the gate will be created and drawn in the canvas using xor operation. The newly created object is held by the variable *currentGate*.

If the active command is to move a gate, *start_rubber_band* will at first try to find the gate to be moved according to the location where the mouse is clicked. If such a gate is found, currentGate is set to the gate and the moving activity continues when the user moves the mouse. If the system can not find such a gate, the moving

activity is aborted by setting currentGate to NULL so that the *track_rubber_band* and *end_rubber_band* will ignore the moving command.

If the active command is to delete a gate, *start_rubber_band* will also at first try to find the gate to be deleted according to the mouse clicking position. If such a gate is found, the gate will be deleted. After the gate is deleted, all wires going into the gate or coming out of the gate will also be deleted. If there is no gate around the mouse clicking position, the deleting activity is simply aborted. Note that in the deleting gate activity, *track_rubber_band* and *end_rubber_band* are not involved because only one mouse click is required.

If the active command is to connect or disconnect two pins in two different gates, the *start_rubber_band* will try to find which pin of which gate the user is interested in by checking the pin location and the mouse clicking position. If the pin can not be found, the activity is aborted. Otherwise, a temporary wire is created and is drawn in the xor graphic context. This temporary wire will be added to the database when the wire is completed after the second pin is selected, or it will be matched to an existing wire to be deleted.

Another possible activity is report the position of a gate. When such a command is active, the *start_rubber_band* will do the same as it processes the deleting gate activity except that after the gate is found, its position is reported in the text widget rather than the deletion of the gate.

After the mouse button is pressed, the user may move the mouse while holding the button. Each small move of the mouse will generate an X-event that will trigger the *track_rubber_band* function. Some of the drawing commands have nothing to do with the *track_rubber_band*, such as position reporting and deleting a gate. Those drawing activities that require two positions take the current mouse position as the new values of the second positions. Therefore, the *track_rubber_band* erases the old temporary drawing and make the new temporary drawing reflecting the new mouse

position.

The important task that the *track_rubber_band* must perform is to keep consistency among gates and wires. Whenever a gate is moved, all related wires must be redrawn in order to let users know the resulting drawing before the activity is done. All involved gates and wires are expected to be made temporary by the *start_rubber_band*, the *track_rubber_band* will erase all those temporary drawing and redraw them to reflect the moving.

The *end_rubber_band* completes the two position drawing activities. First, it will erase the temporary drawing and make a permanent drawing reflecting the current mouse position. Then it will update the database to maintain the consistency with the drawing, if necessary.

Another callback that may be ignored by users is the $refresh$. Since the application program is responsible for restoring the window when its window is exposed again after being overlapped by other windows, our system must be able to draw all the gates and the wires at any time when it is exposed. This callback is $refresh$ which draws all gates and wires at a time. Users may also be able to explicitly ask the system to redraw all components because sometimes the system may not be able to maintain the consistency due the consideration of efficiency and the responding time. This system lacks the feature to detect the overlap of components. Thus, when an overlapped component is erased, the corresponding pixels of another component are also erased.

The simpliest widget is the text widget. It is the system's output window. Because no user input is allowed in this window, there are no callbacks associated with it in the processing level. (Though there are callbacks in the lower level, it is not our concern because Motif handles it for us.) The only function dealing with the text widget is the $PrMsg()$ which simply appends a message to the widget. Though it is simple, more features can be added to the text widget if we expect a more fancy

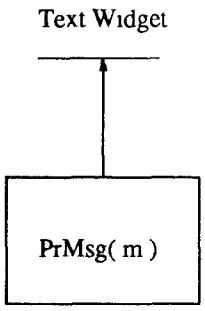

Figure 4.5: Associate Callbacks with Text Widget

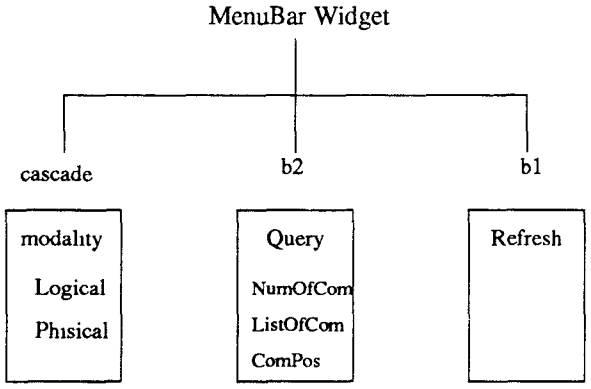

Figure 4.6: Associate Callbacks with Menubar Widget

output.

There is another widget called menubar which allows users to change the modality of the drawing, query the drawing and perform other miscellaneous function about the drawing. Functionally, the menubar consists of all non-drawing functions. It contains several pull down menus having independent functions.

The first pull down menu is the modality selection. It controls the level of the displayed layout. Two levels are provided in this version, gate level (logic circuit) and chip level (physical circuit). We expect to have more levels, such as board level and analog level, when this system is further developed.

The important menu is the query menu, which allows users to get information about the gates and connection. In this version, we implemented some simple queries such as the number of components and the position of a component. Idealy, this is the gateway to the database system. It should be connected to the database in the later version.

4.3 C++ Methods

There are six methods for a gate object. The first one is drawpic(), which draws the picture of the gate. Because every kind of gate has its own appearance, the drawpic() is virtual for gate class. By virtual, we mean that derived classes from gate class must define the drawpic() to meet their own requirement. For the callbacks and other part of the system, all gates are treated as a generic gate. The virtual method drawpic() is called without knowing what kind of gate it is. That is an advantage of the object-oriented programming.

The next pair of methods are also virtual. They are InPos() and OutPos() which return the position of input pins and the output pin. They are both virtual because different kinds of gates have different distribution of pins. Positions of pins are derived from the position of the gate. This pair of methods are used to detect the mouse click. When a mouse click occurs close enough to the position of a pin, we say that this pin is selected.

Another position method is the position() which returns the position of the gate. This method is included to provide read-only feature of the gate position variable.

Other two methods are settle() and move()[1]. They are the same for all gates

[1]Note here that this function deals with the object gate only without considering its relation with other gates. It is mentioned that wires will be automaticly redrawn to keep consistent with the gates they are connected to. Since this feature does not belongs to any class, we do not encapsulate

and so are not virtual. They provide a general service for moving and settling a gate when the gate is being drawn.

When a gate is created, a temporary picture is drawn in the xor graphic context. move() takes a new position, erases the old temporary picture by drawing it again in the xor graphic context, then updates the position of the gate and draws it in the new position, and also in the xor graphic context.

When a user decides to put down a gate, the system will call settle() to make the picture permanent by erasing the old temporary picture and drawing it in the permanent graphic context.

Concrete gates inherit all methods from the class gate. However, they all redefine InPos(), OutPos() and drawpic() to reflect their own appearance.

New kinds of gates can be added by simply defining the InPos(), OutPos() and drawpic().

Wires are a collection of objects that belongs to a single class. The methods of a wire class include drawpic(), drawto(), move(), settle(), deletefrom() and deleteto().

As discussed, a piece of wire involves two pins. When a wire is being drawn, only one pin is determined, we call the wire with one undefined pin a pending wire. A pending wire uses the current mouse cursor position as its second end.

The method drawpic() draws a wire according to two positions, one of which is of a pin and the other is either another pin or a temporary position taken from the current mouse cursor position when the wire is still pending.

The method drawTo() creates a pending wire using a mouse cursor position.

The method move()[2] updates a pending wire by erasing the old pending wire, creating a new one using the new mouse cursor position. All drawing are done in xor graphic context.

---

it in any object, instead, a function MoveGate() is designed to perform this function.

[2]Strictly speaking, it is wire::move() in order to differentiate it from the gate ·move().

The process of method settle() is a little bit different from the process of creating a gate. When a user wants to complete a connection, the system calls wire::settle() in order to settle it. However, the connection may not be a good one since the pending end of a wire may not indicate a pin. Therefore, the task of settle() is to check which pin is close enough to the pending end of a pending wire. If such a pin can be found, the system completes the connection by assigning that pin to the second pin of the wire and the wire is said to be settled. Otherwise, the wire is no good and a connection is aborted.

CHAPTER 5

CONCLUSION

In this thesis, we discussed a CAD system which is implemented based on the theory of graphical Deep Knowledge in the Object-oriented dual model. This research has shown :

1. Graphical Deep Knowledge can be homogeneously integrated into the VML objected-oriented dual model database system.

2. The introduction of the Graphical Deep Knowledge theory enables us to represent the knowledge about objects for reasoning purposes as well as display purpose.

The original prototype was defined in a relational environment using LISP. The second prototype was implemented in a Smalltalk, objected-oriented environment. Our current prototype is in a C++/Objected-oriented/dual model/database system. We support a much larger circuit board design comparing to previous work. In addition, the X-window system is used for the interface design to make the system more flexible to run in different environment.

APPENDIX A

CIRCUIT DATABASE INTERFACE MAIN DRIVE

```
/*************************************************************************
Author : Jue Wang
Name of File : circuit.h Version : 1
Type of File : C++ Header File
Print Date : May 10, 1993 Last Modification : April 17, 1993
Platform : g++, X11 R4, SunOs, Motif
*************************************************************************/
#include <X11/StringDefs.h>
#include <X11/cursorfont.h>
#include <X11/Intrinsic.h>
#include <X11/Xutil.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Xm/ToggleB.h>
#include <Xm/Label.h>
#include <Xm/CascadeB.h>
#include <Xm/ScrolledW.h>
#include <Xm/Text.h>
#include "libXs.h"

#define MAXOBJECTS 1000
extern void PrMsg( char *m );
struct XY { int x, y; };
typedef struct {
 Widget w;
 GC xorgc, gc, erasegc;
 int current_func, foreground, background;
} graphics_data;

/* Gate dimension */
#define GATE_L -30
#define GATE_W 20
#define DIAMETER GATE_W
#define LEG_LEN 5
```

```
#define RADIAN (DIAMETER ≫ 1)
#define NOT_DIAMETER 5
#define ICONLENGTH 45
#define CONNECT 101
#define MOVEGATE 102
#define DELGATE 103
#define DISCONNECT 104
#define ANDGATE 1
#define ORGATE 2
#define NANDGATE 3
#define NORGATE 4
#define XORGATE 5
#define NOTGATE 6
#define BUFFER 7
#define XNORGATE 8
#define RESISTOR 9
#define POWER_SUPPLY 10
#define CAPACITOR 11
#define RESISTOR_LOG 12
#define PNP_TRANSISTOR 13
#define NPN_TRANSISTOR 14
#define PNP_FET 15
#define NPN_FET 16
#define PHIMOD 21
#define COMPOS 22
#ifdef __cplusplus
extern "C" {
#endif
void activate();
void refresh();
void set_stipple();
void start_rubber_band();
void track_rubber_band();
void end_rubber_band();
#ifdef __cplusplus
};
#endif
```

```
/*************************************************************************
Author : Jue Wang
Name of File : circuit.C Version : 1
Type of File : C++ Functions
Print Date : May 10, 1993 Last Modification : April 17, 1993
Remarks : Main Entry, Interface to X and Motif.
Platform : g++, X11 R4, SunOs, Motif
*************************************************************************/
#include <stream.h>
#include <stdlib.h>
#include <stdio.h>
#include "circuit.h"
#include "gate.h"
#include "analog_component.h"
#include "andgate.h"
#include "orgate.h"
#include "nandgate.h"
#include "norgate.h"
#include "xorgate.h"
#include "notgate.h"
#include "wire.h"
#include "chip.h"
#include "buffer.h"
#include "xnorgate.h"
#include "resistor.h"
#include "power_supply.h"
#include "capacitor.h"
#include "resistor_log.h"
#include "pnp_transistor.h"
#include "npn_transistor.h"
#include "pnp_fet.h"
#include "npn_fet.h"

/*************************************************************************
 * Global variable declarations *
 *************************************************************************/
String Resource[] = {
 "draw*XmScrolledWindow*scrollingPolicy:  automatic",
 "draw*XmScrolledWindow*scrollBarDisplayPolicy:  static",
 NULL
};
```

```
Widget msg;
graphics_data data;
gate *gates[MAXOBJECTS];
int numGates, numPins;
void DeleteGate( gate *g );
void DeleteWire( wire *w );
wire *wires[MAXOBJECTS];
int numWires;
gate *currentGate = NULL;
wire *currentWire = NULL;
Chip *currentChip = NULL;

/*********************************************************************
 * A function for printing message *
 *********************************************************************/
void PrMsg( char *m )
{
 XmTextPosition p = XmTextGetLastPosition( msg );
 XmTextSetInsertionPosition( msg, p );
 XmTextInsert( msg, p, m );
 p = XmTextGetLastPosition( msg );
 XmTextSetInsertionPosition( msg, p );
 XmTextInsert( msg, p, "\n" );
 XtSetSensitive( msg, True );
}

/*********************************************************************
 * A function for deciding near a point *
 *********************************************************************/
CloseEnough( int x, int y, XY p )
{
 if( (abs( p.x - x ) < 5) && (abs( p.y - y ) < 5) ) return 1;
 return 0;
}

/*********************************************************************
 * A function for deciding icon area *
 *********************************************************************/
CloseEnough1( int x, int y, XY p )
{
 if( (abs( p.x - x ) < ICONLENGTH) && (abs( p.y - y ) < ICONLENGTH
      )) return 1;
```

```
 return 0;
}

/*****************************************************************
 * Command Widget *
 *****************************************************************/
struct {
 char *name;
 int func;
} command_info[] = {
   "Draw And Gate", ANDGATE,
   "Draw Or Gate", ORGATE,
   "Draw NAnd Gate", NANDGATE,
   "Draw NOr Gate", NORGATE,
   "Draw XOr Gate", XORGATE,
   "Draw Not Gate", NOTGATE,
   "Draw Xnor Gate", XNORGATE,
   "Draw Buffer", BUFFER,
   "Draw Resistor", RESISTOR,
   "Draw Power Supply", POWER_SUPPLY,
   "Draw Capacitor", CAPACITOR,
   "Draw Resistor Log", RESISTOR_LOG,
   "Draw PNP Transistor", PNP_TRANSISTOR,
   "Draw NPN Transistor", NPN_TRANSISTOR,
   "Draw PNP Fet", PNP_FET,
   "Draw NPN Fet", NPN_FET,
   "Connect", CONNECT,
   "Disconnect", DISCONNECT,
   "Move Component", MOVEGATE,
   "Delete Component", DELGATE,
};

void create_drawing_commands( Widget parent, graphics_data *data)
{
 Widget w, commands;
 Arg wargs[2];
 int i;

 /*****************************************
  * Group all commands in a column. *
  *****************************************/
 XtSetArg(wargs[0], XmNentryClass, xmToggleButtonWidgetClass);
```

```
  commands = XmCreateRadioBox(parent, "commands",
                    wargs, 1);
 XtManageChild(commands);

 /***********************************************
  * Create a button for each drawing function. *
  ***********************************************/
 for(i=0;i < XtNumber(command_info); i++){
  XtSetArg(wargs[0], XmNuserData, command_info[i].func );
  w = XtCreateManagedWidget(command_info[i].name,
                    xmToggleButtonWidgetClass,
                    commands, wargs, 1);
  XtAddCallback(w, XmNvalueChangedCallback, activate, data);
 }
};

/*****************************************************
 * initialize data *
 *****************************************************/
void init_data(Widget w, graphics_data *data)
{
 XGCValues values;
 Arg wargs[5];
 data→current_func = 0;
 numGates = numWires = numPins = 0;

 data→w = w;

 /*************************************************
  * Get the colors the user has set for the widget. *
  *************************************************/
 XtSetArg(wargs[0], XtNforeground, &data→foreground);
 XtSetArg(wargs[1], XtNbackground, &data→background);
 XtGetValues(w, wargs,2);

 /*************************************************
  * Fill in the values structure *
  *************************************************/
 values.foreground = data→foreground;
 values.background = data→background;
 values.fill_style = FillTiled;
```

```
 /*************************************************
  * Get the GC used for drawing. *
  *************************************************/
 data→gc= XtGetGC(w, GCForeground | GCBackground |
             GCFillStyle, &values);

 /*****************************************************
  * Get a second GC in XOR mode for rubber banding. *
  *****************************************************/
 data→xorgc = xs_create_xor_gc(w);
 values.foreground = data→background;
 values.background = data→background;
 values.fill_style = FillTiled;

 /*****************************************************
  * Get the GC used for erasing. *
  *****************************************************/
 data→erasegc = XtGetGC(w, GCForeground | GCBackground |
             GCFillStyle, &values);
}

/*************************************************************
 * Activate any function when click middle button *
 *************************************************************/
#define chip_W 50
#define chip_H 100
#define pin_L 10
void start_rubber_band( Widget w, graphics_data *data, XEvent *event)
{
    int x = event→xbutton.x, y = event→xbutton.y;

    switch ( data→current_func ) {
    case ANDGATE:
        currentGate = new AndGate( x, y );
        break;
    case ORGATE:
        currentGate = new OrGate( x, y );
        break;
    case NANDGATE:
        currentGate = new NAndGate( x, y );
        break;
    case XORGATE:
```

```
        currentGate = new XOrGate( x, y );
        break;
    case NORGATE:
        currentGate = new NOrGate( x, y );
        break;
    case NOTGATE:
        currentGate = new NotGate( x, y );
        break;
    case XNORGATE:
        currentGate = new XnOrGate( x, y );
        break;
    case BUFFER:
        currentGate = new Buffer( x, y );
        break;
    case RESISTOR:
        currentGate = new Resistor( x, y );
        break;
    case POWER_SUPPLY:
        currentGate = new Powersupply( x, y );
        break;
    case CAPACITOR:
        currentGate = new Capacitor( x, y );
        break;
    case RESISTOR_LOG:
        currentGate = new Resistorlogical( x, y );
        break;
    case PNP_TRANSISTOR:
        currentGate = new Transistorpnp( x, y );
        break;
    case NPN_TRANSISTOR:
        currentGate = new Transistornpn( x, y );
        break;
    case PNP_FET:
        currentGate = new Fetpnp( x, y );
        break;
    case NPN_FET:
        currentGate = new Fetnpn( x, y );
        break;
    case CONNECT: {
        int i, p;
        gate *g = NULL;
        for(i=0;i<numGates;i++) {
```

```
                g = gates[i];
                if( CloseEnough( x, y, g→InPos(1) ) ) {
                    p = 1;
                    break;
                }
                if( CloseEnough( x, y, g→InPos(2) ) ) {
                    p = 2;
                    break;
                }
                if( CloseEnough( x, y, g→OutPos() ) ) {
                    p = 0;
                    break;
                }
                g = NULL;
            }
            if( g ) {
                currentWire = new wire( g, p );
                currentWire→drawTo( x, y );
                PrMsg( "One pin was selected.  Choose another one please."
);
            } else PrMsg( "No pins was selected." );
            return; }
        case DISCONNECT: {
          int i, p;
          gate *g1 = NULL;
          for(i=0;i<numGates;i++) {
              g1 = gates[i];
              if( CloseEnough( x, y, g1→InPos(1) ) ) {
                  p = 1;
                  break;
              }
              if( CloseEnough( x, y, g1→InPos(2) ) ) {
                  p = 2;
                  break;
              }
              if( CloseEnough( x, y, g1→OutPos() ) ) {
                  p = 0;
                  break;
              }
              g1 = NULL;
          }
        if( g1 ) {
```

```
        currentWire = new wire( g1, p);
        currentWire→drawTo( x, y);
      PrMsg( "One pin was selected.  Choose another one please." );
    } else PrMsg( "No pins was selected." );
    return; }
    case MOVEGATE: {
        int i, p;
        XY pp;
        char buf[80];
        gate *g = NULL;
        for(i=0;i<numGates;i++) {
            g = gates[i];
            // a gate is spcified by the center.

            pp = g→position();
            pp.x += GATE_L/2;
            pp.y += GATE_W/2;
            if( CloseEnough1( x, y, pp ) ) break;
            g = NULL;
        }
        if( g ) {
            currentGate = g;
            PrMsg( "One gate was selected to be moved." );
            StartMovingGate();
        } else {
            PrMsg( "No gates were selected." );
        }
        return; }
    case DELGATE: {
        int i, p;
        XY pp;
        gate *g = NULL;
        for(i=0;i<numGates;i++) {
            g = gates[i];
            // a gate is spcified by the center.

            pp = g→position();
            pp.x += GATE_L/2;
            pp.y += GATE_W/2;
            if( CloseEnough1( x, y, pp ) ) break;
            g = NULL;
        }
```

```
        if( g ) {
            DeleteGate( g );
        } else {
            PrMsg( "No gates were deleted." );
        }
        return; }
    case COMPOS: {
        int i, p;
        char buf[80];
        XY pp;

        gate *g = NULL;
        for(i=0;i<numGates;i++) {
            g = gates[i];
            // a gate is spcified by the center.

            pp = g→position();
            pp.x += GATE_L/2;
            pp.y += GATE_W/2;
            if( CloseEnough1( x, y, pp ) ) break;
            g = NULL;
        }
        if( g ) {
            PrMsg( "The position of this gate is:" );
            pp = g→position();
            sprintf( buf, " ( %d, %d )", pp.x, pp.y);
            PrMsg( buf );
        } else {
            PrMsg( "No gates were selected." );
        }
        return; }
    default: return;
    }
    currentGate→drawpic( data→xorgc );
}

#undef chip_W
#undef chip_H

/*****************************************************************
 * Delete Gate and relevent wires *
 *****************************************************************/
```

```
void DeleteGate( gate *g )
{
    int i;
    g→drawpic( data.erasegc );
    for(i=0;i<numGates;i++) if( g == gates[i] ) break;
    numGates--;
    for( ; i < numGates; i++ ) gates[ i ] = gates[ i + 1 ];
    for( i = 0; i < numWires; i++ )
        if( wires[ i ]→g1 == g || wires[ i ]→g2 == g ) {
            DeleteWire( wires[ i ] );
            i--;
        }
}

/*****************************************************************
 * Delete Wires *
 *****************************************************************/
void DeleteWire( wire *w )
{
    int i;

    for(i=0;i<numWires;i++) if( w == wires[i] ) break;
    numWires--;
    for( ; i < numWires; i++ ) wires[ i ] = wires[ i + 1 ];
    w→drawpic( data.erasegc );
}

/*****************************************************************
 * Start Moving Gate *
 *****************************************************************/
void StartMovingGate()
{
    int i;
    gate *g = currentGate;

    g→drawpic( data.erasegc );
    g→drawpic( data.xorgc );
    for( i = 0; i < numWires; i++ )
        if( wires[ i ]→g1 == g || wires[ i ]→g2 == g ) {
            wires[ i ]→drawpic( data.erasegc );
            wires[ i ]→drawpic( data.xorgc );
        }
```

```
}

/*********************************************************************
 * Move Gate *
 *********************************************************************/
void MoveGate( int x, int y )
{
    int i;
    gate *g = currentGate;

    for( i = 0; i < numWires; i++ )
        if( wires[ i ]→g1 == g || wires[ i ]→g2 == g )
            wires[ i ]→drawpic( data.xorgc );
    g→move( x, y );
    for( i = 0; i < numWires; i++ )
        if( wires[ i ]→g1 == g || wires[ i ]→g2 == g )
            wires[ i ]→drawpic( data.xorgc );
}

/*********************************************************************
 * Stop Move Gate *
 *********************************************************************/
void StopMovingGate()
{
    int i;
    gate *g = currentGate;

    g→drawpic( data.xorgc );
    g→drawpic( data.gc );
    for( i = 0; i < numWires; i++ )
        if( wires[ i ]→g1 == g || wires[ i ]→g2 == g ) {
            wires[ i ]→drawpic( data.xorgc );
            wires[ i ]→drawpic( data.gc );
        }
}

/*********************************************************************
 * Moving mouse when puch the middle button *
 *********************************************************************/
void track_rubber_band( Widget w, graphics_data *data, XEvent *event)
{
    if( data→current_func == MOVEGATE ) {
```

```
        if( currentGate )
            MoveGate( event→xbutton.x, event→xbutton.y );
        return;
    }
    if( data→current_func == PHIMOD ) return;
    if( data→current_func == COMPOS ) return;
    if( data→current_func == DELGATE ) return;
    if( data→current_func == CONNECT ) {
        if( currentWire )
            currentWire→move( event→xbutton.x, event→xbutton.y );
        return;
    }
    if( data→current_func == DISCONNECT ) {
      if( currentWire )
        currentWire→move( event→xbutton.x, event→xbutton.y );
      return;
    }

    // For the drawing objects

    if( data→current_func && currentGate ) {
        currentGate→move( event→xbutton.x, event→xbutton.y );
        return;
    }
}

/*********************************************************************
 * Store gate *
 *********************************************************************/
void store_object()
{
 /***********************************************
  * Check for space. *
  ***********************************************/
 if(numGates ≥ MAXOBJECTS){
 cout ≪ "Warning:  Graphics buffer is full\n";
 return;
 }

 /*****************************************************
  * Save everything we need to draw this object again. *
  *****************************************************/
```

```
 gates[numGates] = currentGate;
 currentGate = NULL;

 /*****************************************
  * Increment the next position index. *
  *****************************************/
 numGates++;
}

/************************************************************
 * store wire *
 ************************************************************/
void store_wire()
{
 /*****************************
  * Check for space. *
  *****************************/
 if(numWires ≥ MAXOBJECTS){
 cout ≪ "Warning:  Graphics buffer is full\n";
 return;
 }

 /*******************************************************
  * Save everything we need to draw this object again. *
  *******************************************************/
 wires[numWires] = currentWire;
 currentWire = NULL;

 /***************************************
  * Increment the next position index. *
  ***************************************/
 numWires++;
}

void delete_wire()
{
     int i;
     wire *w;

     for(i=0;i<numWires;i++) {
          if( currentWire→g1 == wires[i]→g1 &&
            currentWire→g2 == wires[i]→g2 &&
```

```
            currentWire→pin1 == wires[i]→pin1 &&
            currentWire→pin2 == wires[i]→pin2 ||
            currentWire→g1 == wires[i]→g2 &&
            currentWire→g2 == wires[i]→g1 &&
            currentWire→pin1 == wires[i]→pin2 &&
            currentWire→pin2 == wires[i]→pin1 ) {
                w = wires[i];
                DeleteWire( w );
                PrMsg( "Wire was deleted." );
                return;
            }
        }
        currentWire→drawpic( data.erasegc );
        PrMsg( "Wire does not exist\n" );
}

/*************************************************************
 * stop clicking middle button *
 *************************************************************/
void end_rubber_band( Widget w, graphics_data *data, XEvent *event)
{
    if( data→current_func == MOVEGATE ) {
        if( currentGate ) {
            StopMovingGate();
            currentGate = NULL;
        }
        return;
    }
    if( data→current_func == CONNECT ) {
        if( currentWire ) {
            if( currentWire→settle() ) {
                store_wire();
                PrMsg( "Wire was created." );
            } else {
                delete currentWire;
                PrMsg( "No pins were selected, wire was deleted." );
            }
            currentWire = NULL;
        }
        return;
    }
    if( data→current_func == DISCONNECT ) {
```

```
        if( currentWire ) {
            if( currentWire→settle() ) delete_wire();
            delete currentWire;
            currentWire = NULL;
        }
        return;
    }
    if( data→current_func == PHIMOD ) return;
    if( data→current_func == DELGATE ) return;
    if( data→current_func == COMPOS ) return;
    if( data→current_func && currentGate ) {
        currentGate→settle();
        store_object();
    }
}

/*************************************************************
 * Activate function for XtAddCallback *
 *************************************************************/
void activate(Widget w, graphics_data *data,
        XmToggleButtonCallbackStruct *call_data)
{
 int func;
 Arg wargs[5];

 if(!call_data→set) return;
 XtSetArg(wargs[0], XmNuserData, &func);
 XtGetValues(w, wargs, 1);
 data→current_func = func;
}

/*************************************************************
 * redraw picture *
 *************************************************************/
void refresh(Widget w, graphics_data *data)
{
 int i;
 for(i=0;i<numGates;i++) gates[i]→drawpic( data→gc );
 for(i=0;i<numWires;i++) wires[i]→drawpic( data→gc );
 if (! currentChip == NULL)
 currentChip→drawpic( data→gc );
}
```

```
/*****************************************************************
 * erase gates *
 *****************************************************************/
void erasegates(Widget w, graphics_data *data)
{
 int i;

 for(i=0;i<numGates;i++) gates[i]→drawpic( data→erasegc );
}

/*****************************************************************
 * erase wire *
 *****************************************************************/
void erasewires(Widget w, graphics_data *data)
{
 int i;

 for(i=0;i<numWires;i++) wires[i]→drawpic( data→erasegc );
}

/*****************************************************************
 * erase chip *
 *****************************************************************/
void erasechip(Widget w, graphics_data *data)
{
 if( !currentChip == NULL ) currentChip→drawpic(data→erasegc );
}

/*****************************************************************
 * erase all the objects *
 *****************************************************************/
void eraseall(Widget w, graphics_data *data)
{
 erasegates( w, data );
 erasewires( w, data );
 erasechip( w, data );
}

/*****************************************************************
 * Query for List of Components *
 *****************************************************************/
```

```
void Query()
{
    char buf[80];
    gate *g;
    wire *w;
    char *type;
    int i;

    PrMsg( "----------------- Gate Information --------------------" );
    sprintf( buf, " GateNo.  Type Pos" );
    PrMsg( "-----------------------------------" );
    PrMsg( buf );
    for( i = 0; i < numGates; i++ ) {
        g = gates[ i ];
        switch ( g→type ) {
        case ANDGATE: type = "AND"; break;
        case ORGATE: type = "OR"; break;
        case NANDGATE: type = "NAND"; break;
        case XORGATE: type = "XOR"; break;
        case NORGATE: type = "NOR"; break;
        case NOTGATE: type = "NOT"; break;
        case BUFFER: type = "BUFFER"; break;
        case XNORGATE: type = "XNOR"; break;
        }
        sprintf( buf, " %4d %4s ( %d, %d )", i + 1, type, g→pos.x, g→pos.y );
        PrMsg( buf );
    }
    PrMsg( "\n" );
    PrMsg( " Connection" );
    sprintf( buf, " Gate-Pin 1 Gate-Pin 2" );
    PrMsg( buf );
    for( i = 0; i < numWires; i++ ) {
        int g1, g2, j;
        w = wires[i];
        for( j = 0; j<numGates; j++ ) {
            if( gates[j] == w→g1 ) g1 = j + 1;
            if( gates[j] == w→g2 ) g2 = j + 1;
        }
        sprintf( buf, " <%d,%d> <%d,%d>", g1, w→pin1, g2, w→pin2 );
        PrMsg( buf );
    }
}
```

```
/*****************************************************************
 * Print the number of gates *
 *****************************************************************/
void NumOfGates ()
{
    char buf[50];

    PrMsg( "The number of components is:");
    sprintf( buf, " %d ", numGates );
    PrMsg( buf );
}

/*****************************************************************
 * Modility Menu *
 *****************************************************************/
Widget make_menu2_option(char *option_name, void (*func)(Widget,
graphics_data *), Widget menu2)
{
 int n;
 Arg wargs[10];
 Widget b2;

 n = 0;
 XtSetArg(wargs[n], XmNlabelString,
    XmStringCreateLtoR(option_name,
    XmSTRING_DEFAULT_CHARSET)); n++;
 b2=XtCreateManagedWidget(option_name,xmPushButtonWidgetClass,
    menu2,wargs,n);
 XtAddCallback (b2, XmNactivateCallback, func, &data);
 XtSetSensitive(b2, True);
 return(b2);
}

Widget make_menu2(char *menu2_name, Widget menuBar)
{
 int n;
 Arg wargs[10];
 Widget menu2, cascade;

 n = 0;
 menu2 = XmCreatePulldownMenu (menuBar, menu2_name, wargs, n);
```

```
 n = 0;
 XtSetArg (wargs[n], XmNsubMenuId, menu2); n++;
 XtSetArg(wargs[n], XmNlabelString,
 XmStringCreateLtoR(menu2_name, XmSTRING_DEFAULT_CHARSET)); n++;
 cascade = XmCreateCascadeButton (menuBar, menu2_name, wargs, n);
 XtManageChild (cascade);
 return(menu2);
}

void logmod ()
{
    char buf[100];

    sprintf( buf, "You can draw any circuit board by clicking some
    categories on the left column");
    PrMsg( buf );
}

void PhiMod(Widget w, graphics_data *data)
{
        int i, t, edges;

        eraseall(w, data);
        t = 0;
        for(i=0;i<numGates;i++)
         if ( gates[i]->type == NOTGATE ) t++;
        edges = ( 3*numGates - t + 3 ) / 2;
        currentChip = new Chip(edges);
        currentChip->drawpic( data->gc );
}

void logical(Widget w, graphics_data *data)
{
 int i;

 erasechip(w, data);
 for(i=0;i<numGates;i++) gates[i]->drawpic( data->gc );
 for(i=0;i<numWires;i++) wires[i]->drawpic( data->gc );
}

void create_menus2(Widget menuBar)
{
```

```
 int n;
 Arg wargs[10];
 Widget menu2, b;
 void Query();

 menu2=make_menu2("Modility ",menuBar);
 make_menu2_option("Logical",logical, menu2);
 make_menu2_option("Phyical",PhiMod, menu2);
}

void create_menus3(Widget menuBar)
{
 int n;
 Arg wargs[10];
 Widget menu2, b;
 void Query();

 menu2=make_menu2("Refresh ",menuBar);
 make_menu2_option("Refresh",refresh, menu2);
}

/*************************************************************
 * Query Menu *
 *************************************************************/
void ComPos ()
{
     data.current_func = COMPOS;
}

Widget make_menu_option(char *option_name, void (*func)(), Widget menu )
{
 int n;
 Arg wargs[10];
 Widget b;

 n = 0;
 XtSetArg(wargs[n], XmNlabelString,
     XmStringCreateLtoR(option_name,
     XmSTRING_DEFAULT_CHARSET)); n++;
 b=XtCreateManagedWidget(option_name,xmPushButtonWidgetClass,
     menu,wargs,n);
 XtAddCallback (b, ( const char* )XmNactivateCallback, func, &data);
```

```
 XtSetSensitive(b, True);
 return(b);
}

Widget make_menu(char *menu_name, Widget menuBar)
{
 int n;
 Arg wargs[10];
 Widget menu, cascade;

 n = 0;
 menu = XmCreatePulldownMenu (menuBar, menu_name, wargs, n);
 n = 0;
 XtSetArg (wargs[n], XmNsubMenuId, menu); n++;
 XtSetArg(wargs[n], XmNlabelString,
 XmStringCreateLtoR(menu_name, XmSTRING_DEFAULT_CHARSET)); n++;
 cascade = XmCreateCascadeButton (menuBar, menu_name, wargs, n);
 XtManageChild (cascade);
 return(menu);
}

void create_menus(Widget menuBar)
{
 int n;
 Arg wargs[10];
 Widget menu, b;
 void Query();

 menu=make_menu("Query ",menuBar);
 make_menu_option("NumberOfComponents", NumOfGates, menu);
 make_menu_option("ListOfComponents", Query, menu);
 make_menu_option("ComponentPosition", ComPos, menu);
}

#define HEIGHT 800
#define WIDTH 1000
/*******************************************************
 * Main function *
 *******************************************************/
#if (XlibSpecificationRelease≥5)
void main ( int argc, char **argv )
#else
```

```
void main ( unsigned int argc, char **argv )
#endif
{
 Widget toplevel, canvas, framework, command, tiles,
        sw, menuBar;
 int n;
 Arg wargs[10];
 XtAppContext app;

/*****************************************************
 * Create a overall framework *
 *****************************************************/
 toplevel = XtAppInitialize ( &app, "draw", NULL, 0,
                &argc, argv, Resource, NULL, 0 );
 framework = XtCreateManagedWidget("framework",
                      xmFormWidgetClass,
                      toplevel, NULL, 0);
 n = 0;
 XtSetArg(wargs[n], XmNheight, HEIGHT ); n++;
 XtSetArg(wargs[n], XmNwidth, WIDTH ); n++;
 XtSetValues(framework, wargs, n);

/*****************************************************
 * Create the menu bar *
 *****************************************************/
 n=0;
 menuBar=XmCreateMenuBar(framework,"menuBar",wargs,n);
 XtManageChild(menuBar);

/*****************************************************
 * Attach the menu bar to the form *
 *****************************************************/
 n=0;
 XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_FORM); n++;
 XtSetArg(wargs[n], XmNrightAttachment, XmATTACH_FORM); n++;
 XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_FORM); n++;
 XtSetValues(menuBar,wargs,n);

/*****************************************************
 * Attach submenus to the menuBar *
 *****************************************************/
 create_menus2(menuBar);
```

```
create_menus(menuBar);
create_menus3(menuBar);

/*************************************************
 * Create the column to hold the commands. *
 *************************************************/
command = XtCreateManagedWidget("command",
              xmRowColumnWidgetClass,
              framework, NULL, 0);
n = 0;
XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_WIDGET);n++;
XtSetArg(wargs[n], XmNtopWidget, menuBar);n++;
XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_FORM);n++;
XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_FORM);n++;
XtSetValues(command, wargs, n);

/*************************************************
* create a text widget *
*************************************************/
n = 0;
XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_FORM);n++;
XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_WIDGET);n++;
XtSetArg(wargs[n], XmNleftWidget, command); n++;
XtSetArg(wargs[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(wargs[n], XmNeditMode, XmMULTI_LINE_EDIT ); n++;
XtSetArg(wargs[n], XmNheight, HEIGHT/4 ); n++;
XtSetArg(wargs[n], XmNwidth, WIDTH/2 ); n++;
msg = XmCreateScrolledText( framework, "msg", wargs, n );
XtManageChild( msg );
XtSetSensitive( msg, False );
XmTextSetAddMode( msg, True );
XmTextSetEditable( msg, False );
PrMsg( "Welcome to circuit world.  You can read message here." );

//*******************************************************

// A ScrolledWindow widget occupies the bottom portion

// of the window, and spans the entire width

//*******************************************************
```

```
sw = XtVaCreateManagedWidget ( "sw",
      xmScrolledWindowWidgetClass,
      framework, NULL );
n = 0;
XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_WIDGET);n++;
XtSetArg(wargs[n], XmNtopWidget, menuBar);n++;
XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_WIDGET); n++;
XtSetArg(wargs[n], XmNbottomWidget, msg); n++;
XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_WIDGET);n++;
XtSetArg(wargs[n], XmNleftWidget, command); n++;
XtSetArg(wargs[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetValues(sw, wargs, n);

// A DrawingArea widget provides a scrollable work area

/*****************************************************
 * Create the drawing surface and add the *
 * rubber banding callbacks. *
 *****************************************************/
canvas = XtCreateManagedWidget("canvas",
              xmDrawingAreaWidgetClass,
              sw, NULL, 0);
XtAddCallback(canvas, XmNexposeCallback, refresh, &data);
XtAddEventHandler(canvas, ButtonPressMask, FALSE,
        start_rubber_band, &data ) ;
XtAddEventHandler(canvas, ButtonMotionMask, FALSE,
           track_rubber_band, &data);
XtAddEventHandler(canvas, ButtonReleaseMask, FALSE,
           end_rubber_band, &data);
n = 0;
XtSetArg(wargs[n], XmNheight, HEIGHT*2 ); n++;
XtSetArg(wargs[n], XmNwidth, WIDTH*2 ); n++;
XtSetValues(canvas, wargs, n);

/*****************************************************
 * Initialize the graphics buffer and other data. *
 *****************************************************/
init_data(canvas, &data);

/*****************************************************
 * Add a quit button. *
 *****************************************************/
```

```
xs_create_quit_button(command);

/*****************************************************
 * Add the drawing command panel. *
 *****************************************************/
create_drawing_commands(command, &data);
XtRealizeWidget( toplevel );

/*********************************************************
 * Establish a passive grab on the drawing canvas window.*
 ********************************************************* */
XGrabButton(XtDisplay(canvas), AnyButton, AnyModifier,
        XtWindow(canvas), TRUE,
        ButtonPressMask | ButtonMotionMask |
        ButtonReleaseMask,
        GrabModeAsync, GrabModeAsync,
        XtWindow(canvas),
        XCreateFontCursor(XtDisplay(canvas),
                XC_hand2));

/*****************************************************
* Set titles *
*****************************************************/
XStoreName(XtDisplay(toplevel),XtWindow(toplevel),"A circuit");
XSetIconName(XtDisplay(toplevel), XtWindow(toplevel),"circuit");
XtAppMainLoop( app );
}
```

APPENDIX B

CIRCUIT SCHEMA

```
SCHEMA CircuitSchema
IMPORT FROM DatatypeCircuitSchema;
IMPORT FROM SemanticSchema;
IMPORT FROM CategorySchema;
IMPORT FROM R_SPEC_C_SPEC_SCHEMA;
DEFINE aspect 1

OBJECTTYPE pc_boardType [
    pc_boardClass : pc_boardType,
      board_chipsClass : board_chipsType,
      digital_circuitsClass : digital_circuitsType,
      analog_circuitsClass : analog_circuitsType,
      junctionsClass : junctionsType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Size : RECTANGLETYPE;
    Title : STRING;
    Board_junctions : junctionsClass;
    Analog_circuits : analog_circuitsClass;
    Digital_circuits : digital_circuitsClass;
    Chips : board_chipsClass;
  IMPLEMENTATION
  METHODS
    method_pc_boardType() READONLY; {};
END;

CLASS pc_board
  INSTTYPE pc_boardType [
    pc_board, board_chips, digital_circuits,
    analog_circuits, junctions ]
END;

OBJECTTYPE chipType [
    chipClass : chipType,
      chipsClass : chipsType,
      digital_circuitsClass : digital_circuitsType,
```

```
      analog_circuitsClass : analog_circuitsType,
      pinsClass : pinsType,
      junctionsClass : junctionsType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Size : RECTANGLETYPE;
    Title : STRING;
    Chip_junctions : junctionsClass;
    Chip_pins : pinsClass;
    Chip_analog_circuits : analog_circuitsClass;
    Chip_digital_circuits : digital_circuitsClass;
    memberof : chipsClass;
  IMPLEMENTATION
  METHODS
    method_chipType() READONLY; {};
END;

CLASS chip
  INSTTYPE chipType [ chip, chips, digital_circuits,
      analog_circuits, pins, junctions ]
END;

OBJECTTYPE chipsType [ chipsClass : chipsType,
                       chipClass : chipType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
               Purpose : STRING;
               Cardinality : INT;
               setof : chipClass;
  IMPLEMENTATION
  METHODS
    method_chipsType() READONLY; {};
END;

CLASS chips
  INSTTYPE chipsType [
    chips, chip ]
END;

OBJECTTYPE board_chipType [
```

```
    board_chipClass : board_chipType,
      board_chipsClass : board_chipsType,
      board_pinsClass : board_pinsType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Position : POINTTYPE;
    Label : STRING;
    Size : RECTANGLETYPE;
    Board_chip_cons : connectionsClass;
    Board_chip_junc : junctionsClass;
    Board_chip_pins : board_pinsClass;
    memberof : board_chipsClass;
  IMPLEMENTATION
  METHODS
    method_board_chipType() READONLY; {};
END;

CLASS board_chip  METACLASS ROLE_SPECIALIZATION_CLASS
  INSTTYPE board_chipType [
    board_chip, board_chips, board_pins, junctions, connections ]
  INIT  board_chip->defRoleClass(  chip  )
END;

OBJECTTYPE board_chipsType [
    board_chipsClass : board_chipsType,
      board_chipClass : board_chipType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
              Purpose : STRING;
              Cardinality : INT;
              setof : board_chipClass;
  IMPLEMENTATION
  METHODS
    method_board_chipsType() READONLY; {};
END;

CLASS board_chips
  INSTTYPE board_chipsType [
```

```
    board_chips, board_chip ]
END;

OBJECTTYPE junctionType [
    junctionClass : junctionType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Position : POINTTYPE;
    Label : STRING;
    Junction_for : connectionsClass;
    memberof : junctionsClass;
  IMPLEMENTATION
  METHODS
    method_junctionType() READONLY; {};
END;

CLASS junction
  INSTTYPE junctionType [
    junction, junctions, connections ]
END;

OBJECTTYPE junctionsType [
    junctionsClass : junctionsType,
      junctionClass : junctionType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
                Purpose : STRING;
                Cardinality : INT;
                setof : junctionClass;
  IMPLEMENTATION
  METHODS
    method_junctionsType() READONLY; {};
END;

CLASS junctions
  INSTTYPE junctionsType [ junctions, junction ]
END;
```

```
OBJECTTYPE connectionType [
    connectionClass : connectionType,
      connectionsClass : connectionsType,
      pinClass : pinType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Position : POINTTYPE;
    Pin2 : pinClass;
    Pin1 : pinClass;
    memberof : connectionsClass;
  IMPLEMENTATION
  METHODS
    method_connectionType() READONLY; {};
END;

CLASS connection
  INSTTYPE connectionType [
    connection, connections, pin ]
END;

OBJECTTYPE connectionsType [
    connectionsClass : connectionsType,
      connectionClass : connectionType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
                Purpose : STRING;
                Cardinality : INT;
                setof : connectionClass;
  IMPLEMENTATION
  METHODS
    method_connectionsType() READONLY; {};
END;

CLASS connections
  INSTTYPE connectionsType [
    connections, connection ]
END;

OBJECTTYPE pinType [
    pinClass : pinType,
```

```
      pinsClass : pinsType,
      componentClass : componentType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Length : INT;
    Label : STRING;
    Side : INT;
    Position : POINTTYPE;
    Pin_of : componentClass;
    memberof : pinsClass;
  IMPLEMENTATION
  METHODS
    method_pinType() READONLY; {};
END;

CLASS pin
  INSTTYPE pinType [ pin, pins, component ]
END;

OBJECTTYPE pinsType [
    pinsClass : pinsType,
      pinClass : pinType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
                Purpose : STRING;
                Cardinality : INT;
                setof : pinClass;
  IMPLEMENTATION
  METHODS
    method_pinsType() READONLY; {};
END;

CLASS pins
  INSTTYPE pinsType [ pins, pin ]
END;

OBJECTTYPE board_pinType [
    board_pinClass : board_pinType,
      board_pinsClass : board_pinsType,
      pc_boardClass : pc_boardType ]
```

```
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Name : STRING;
    Belongs_to : pc_boardClass;
    Attached_to : board_pinsClass;
    memberof : board_pinsClass;
  IMPLEMENTATION
  METHODS
    method_board_pinType() READONLY; {};
END;

CLASS board_pin METACLASS ROLE_SPECIALIZATION_CLASS
  INSTTYPE board_pinType [
    board_pin, board_pins, pc_board ]
  INIT  board_pin->defRoleClass(  pin  )
END;

OBJECTTYPE board_pinsType [
    board_pinsClass : board_pinsType,
      board_pinClass : board_pinType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
                Purpose : STRING;
                Cardinality : INT;
                setof : board_pinClass;
  IMPLEMENTATION
  METHODS
    method_board_pinsType() READONLY; {};
END;

CLASS board_pins
  INSTTYPE board_pinsType [ board_pins, board_pin ]
END;

OBJECTTYPE board_componentType [
    board_componentClass : board_componentType,
      board_componentsClass : board_componentsType,
      connectionsClass : connectionsType,
      pc_boardClass : pc_boardType ]
        SUBTYPEOF Metaclass_InstType;
```

```
  INTERFACE
  PROPERTIES
    Position : POINTTYPE;
    Area : RECTANGLETYPE;
    Component_of : pc_boardClass;
    Component_connected_to : connectionsClass;
    memberof : board_componentsClass;
  IMPLEMENTATION
  METHODS
    method_board_componentType() READONLY; {};
END;

CLASS board_component METACLASS ROLE_SPECIALIZATION_CLASS
  INSTTYPE board_componentType [
    board_component, board_components, connections, pc_board ]
  INIT  board_component->defRoleClass(  component  )
END;

OBJECTTYPE board_componentsType [
    board_componentsClass : board_componentsType,
      board_componentClass : board_componentType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
                Purpose : STRING;
                Cardinality : INT;
                setof : board_componentClass;
  IMPLEMENTATION
  METHODS
    method_board_componentsType() READONLY; {};
END;

CLASS board_components
  INSTTYPE board_componentsType [
    board_components, board_component ]
END;

OBJECTTYPE board_digital_componentType [
    board_digital_componentClass : board_digital_componentType,
      board_digital_componentsClass : board_digital_componentsType,
      board_componentClass : board_componentType,
      board_componentsClass : board_componentsType,
```

```
      connectionsClass : connectionsType,
      pc_boardClass : pc_boardType ]
        SUBTYPEOF board_componentType [
    board_componentClass,
          board_componentsClass,
          connectionsClass,
          pc_boardClass ];
  INTERFACE
  PROPERTIES
    Num_input : INT;
    Num_output : INT;
    Inputs : IN_OUT_SET;
    memberof1 : board_digital_componentsClass;
  IMPLEMENTATION
  METHODS
    method_board_digital_componentType() READONLY; {};
END;

CLASS board_digital_component METACLASS R_SPEC_C_SPEC_CLASS
  INSTTYPE board_digital_componentType [
board_digital_component,board_digital_components,board_component,
board_components,connections,pc_board]
  INIT  board_digital_component->defRoleClass(  digital_component  );
    board_digital_component->defCategory( board_component, aspect  )
END;

OBJECTTYPE board_digital_componentsType [
    board_digital_componentsClass : board_digital_componentsType,
      board_digital_componentClass : board_digital_componentType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
               Purpose : STRING;
               Cardinality : INT;
               setof : board_digital_componentClass;
  IMPLEMENTATION
  METHODS
    method_board_digital_componentsType() READONLY; {};
END;

CLASS board_digital_components
  INSTTYPE board_digital_componentsType [
```

```
    board_digital_components, board_digital_component ]
END;

OBJECTTYPE board_analog_componentType [
    board_analog_componentClass : board_analog_componentType,
      board_analog_componentsClass : board_analog_componentsType,
      board_componentClass : board_componentType,
      board_componentsClass : board_componentsType,
      connectionsClass : connectionsType,
      pc_boardClass : pc_boardType ]
        SUBTYPEOF board_componentType [
    board_componentClass,
          board_componentsClass,
          connectionsClass,
          pc_boardClass ];
  INTERFACE
  PROPERTIES
    Output_Voltage : REAL;
    Input_voltage : REAL;
    memberof1 : board_analog_componentsClass;
  IMPLEMENTATION
  METHODS
    method_board_analog_componentType() READONLY; {};
END;

CLASS board_analog_component  METACLASS R_SPEC_C_SPEC_CLASS
  INSTTYPE board_analog_componentType [
board_analog_component,board_analog_components,board_component,
board_components,connections,pc_board]
  INIT  board_analog_component->defRoleClass(  analog_component  );
    board_analog_component->defCategory( board_component, aspect  )
END;

OBJECTTYPE board_analog_componentsType [
    board_analog_componentsClass : board_analog_componentsType,
      board_analog_componentClass : board_analog_componentType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
               Purpose : STRING;
               Cardinality : INT;
               setof : board_analog_componentClass;
```

```
  IMPLEMENTATION
  METHODS
    method_board_analog_componentsType() READONLY; {};
END;

CLASS board_analog_components
  INSTTYPE board_analog_componentsType [
    board_analog_components, board_analog_component ]
END;

OBJECTTYPE componentType [
    componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Name : STRING;
    Modality : STRING;
    Color : INT;
    Label : STRING;
    Size : INT;
    Pins : pinsClass;
    memberof : componentsClass;
  IMPLEMENTATION
  METHODS
    method_componentType() READONLY; {};
END;

CLASS component
  INSTTYPE componentType [
    component, components, pins ]
END;

OBJECTTYPE componentsType [
    componentsClass : componentsType,
      componentClass : componentType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
                Purpose : STRING;
                Cardinality : INT;
```

```
                setof : componentClass;
  IMPLEMENTATION
  METHODS
    method_componentsType() READONLY; {};
END;

CLASS components
  INSTTYPE componentsType [
    components, component ]
END;

OBJECTTYPE digital_componentType [
    digital_componentClass : digital_componentType,
      digital_componentsClass : digital_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF componentType [
    componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Outputs : INT;
    Inputs : INT;
    memberof1 : digital_componentsClass;
  IMPLEMENTATION
  METHODS
    method_digital_componentType() READONLY; {};
END;

CLASS digital_component METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE digital_componentType [
digital_component,digital_components,component,components,pins]
  INIT  digital_component->defCategory( component, aspect  )
END;

OBJECTTYPE digital_componentsType [
    digital_componentsClass : digital_componentsType,
      digital_componentClass : digital_componentType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
```

```
  PROPERTIES
                Purpose : STRING;
                Cardinality : INT;
                setof : digital_componentClass;
  IMPLEMENTATION
  METHODS
    method_digital_componentsType() READONLY; {};
END;

CLASS digital_components
  INSTTYPE digital_componentsType [
    digital_components, digital_component ]
END;

OBJECTTYPE gateType [
    gateClass : gateType,
      digital_componentClass : digital_componentType,
      digital_componentsClass : digital_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF digital_componentType [
    digital_componentClass,
          digital_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Prop_delay : REAL;
    Num_inputs : INT;
    Noise_margin : REAL;
    Power_dis : REAL;
    Fanout : INT;
  IMPLEMENTATION
  METHODS
    method_gateType() READONLY; {};
END;

CLASS gate  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE gateType [
gate,digital_component,digital_components,component,components,pins]
```

```
  INIT  gate->defCategory( digital_component, aspect  )
END;

OBJECTTYPE bufferType [
    bufferClass : bufferType,
      gateClass : gateType,
      digital_componentClass : digital_componentType,
      digital_componentsClass : digital_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF gateType [
    gateClass,
          digital_componentClass,
          digital_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Function : STRING;
  IMPLEMENTATION
  METHODS
    method_bufferType() READONLY; {};
END;

CLASS buffer  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE bufferType [
buffer,gate,digital_component,digital_components,component,
components,pins]
  INIT  buffer->defCategory( gate, aspect  )
END;

OBJECTTYPE xnor_gateType [
    xnor_gateClass : xnor_gateType,
      gateClass : gateType,
      digital_componentClass : digital_componentType,
      digital_componentsClass : digital_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF gateType [
```

```
    gateClass,
          digital_componentClass,
          digital_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Function : STRING;
  IMPLEMENTATION
  METHODS
    method_xnor_gateType() READONLY; {};
END;

CLASS xnor_gate METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE xnor_gateType [
xnor_gate,gate,digital_component,digital_components,component,
components,pins]
  INIT  xnor_gate->defCategory( gate, aspect  )
END;

OBJECTTYPE not_gateType [
    not_gateClass : not_gateType,
      gateClass : gateType,
      digital_componentClass : digital_componentType,
      digital_componentsClass : digital_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF gateType [
    gateClass,
          digital_componentClass,
          digital_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Special_name : STRING;
    Function : STRING;
  IMPLEMENTATION
  METHODS
```

```
    method_not_gateType() READONLY; {};
END;

CLASS not_gate  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE not_gateType [
not_gate,gate,digital_component,digital_components,component,
components,pins]
  INIT  not_gate->defCategory( gate, aspect  )
END;

OBJECTTYPE xor_gateType [
    xor_gateClass : xor_gateType,
      gateClass : gateType,
      digital_componentClass : digital_componentType,
      digital_componentsClass : digital_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF gateType [
    gateClass,
          digital_componentClass,
          digital_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Function : STRING;
  IMPLEMENTATION
  METHODS
    method_xor_gateType() READONLY; {};
END;

CLASS xor_gate  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE xor_gateType [
xor_gate,gate,digital_component,digital_components,component,
components,pins]
  INIT  xor_gate->defCategory( gate, aspect  )
END;

OBJECTTYPE nor_gateType [
    nor_gateClass : nor_gateType,
```

```
      gateClass : gateType,
      digital_componentClass : digital_componentType,
      digital_componentsClass : digital_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF gateType [
    gateClass,
          digital_componentClass,
          digital_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Function : STRING;
  IMPLEMENTATION
  METHODS
    method_nor_gateType() READONLY; {};
END;

CLASS nor_gate  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE nor_gateType [
nor_gate,gate,digital_component,digital_components,component,
components,pins]
  INIT  nor_gate->defCategory( gate, aspect  )
END;

OBJECTTYPE or_gateType [
    or_gateClass : or_gateType,
      gateClass : gateType,
      digital_componentClass : digital_componentType,
      digital_componentsClass : digital_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF gateType [
    gateClass,
          digital_componentClass,
          digital_componentsClass,
          componentClass,
          componentsClass,
```

```
          pinsClass ];
  INTERFACE
  PROPERTIES
    Function : STRING;
  IMPLEMENTATION
  METHODS
    method_or_gateType() READONLY; {};
END;

CLASS or_gate METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE or_gateType [
or_gate,gate,digital_component,digital_components,component,
components,pins]
  INIT  or_gate->defCategory( gate, aspect  )
END;

OBJECTTYPE nand_gateType [
    nand_gateClass : nand_gateType,
      gateClass : gateType,
      digital_componentClass : digital_componentType,
      digital_componentsClass : digital_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF gateType [
    gateClass,
          digital_componentClass,
          digital_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Function : STRING;
  IMPLEMENTATION
  METHODS
    method_nand_gateType() READONLY; {};
END;

CLASS nand_gate METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE nand_gateType [
nand_gate,gate,digital_component,digital_components,component,
```

```
components,pins]
  INIT  nand_gate->defCategory( gate, aspect  )
END;

OBJECTTYPE and_gateType [
    and_gateClass : and_gateType,
      gateClass : gateType,
      digital_componentClass : digital_componentType,
      digital_componentsClass : digital_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF gateType [
    gateClass,
          digital_componentClass,
          digital_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Function : STRING;
  IMPLEMENTATION
  METHODS
    method_and_gateType() READONLY; {};
END;

CLASS and_gate  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE and_gateType [
and_gate,gate,digital_component,digital_components,component,
components,pins]
  INIT  and_gate->defCategory( gate, aspect  )
END;

OBJECTTYPE analog_componentType [
    analog_componentClass : analog_componentType,
      analog_componentsClass : analog_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF componentType [
    componentClass,
```

```
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Max_current : REAL;
    memberof1 : analog_componentsClass;
  IMPLEMENTATION
  METHODS
    method_analog_componentType() READONLY; {};
END;

CLASS analog_component  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE analog_componentType [
    analog_component, analog_components, component, components, pins ]
  INIT  analog_component->defCategory( component, aspect  )
END;

OBJECTTYPE analog_componentsType [
    analog_componentsClass : analog_componentsType,
      analog_componentClass : analog_componentType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
               Purpose : STRING;
               Cardinality : INT;
               setof : analog_componentClass;
  IMPLEMENTATION
  METHODS
    method_analog_componentsType() READONLY; {};
END;

CLASS analog_components
  INSTTYPE analog_componentsType [
    analog_components, analog_component ]
END;

OBJECTTYPE TransistorType [
    TransistorClass : TransistorType,
      analog_componentClass : analog_componentType,
      analog_componentsClass : analog_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
```

```
      pinsClass : pinsType ]
        SUBTYPEOF analog_componentType [
    analog_componentClass,
          analog_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Cut_off_vol : REAL;
  IMPLEMENTATION
  METHODS
    method_TransistorType() READONLY; {};
END;

CLASS Transistor  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE TransistorType [
Transistor,analog_component,analog_components,component,components,
pins]
  INIT  Transistor->defCategory( analog_component, aspect  )
END;

OBJECTTYPE fetType [
    fetClass : fetType,
      TransistorClass : TransistorType,
      analog_componentClass : analog_componentType,
      analog_componentsClass : analcg_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF TransistorType [
    TransistorClass,
          analog_componentClass,
          analog_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Gate_vol : REAL;
  IMPLEMENTATION
  METHODS
```

```
    method_fetType() READONLY; {};
END;

CLASS fet METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE fetType [
fet,Transistor,analog_component,analog_components,component,
components,pins]
  INIT  fet->defCategory( Transistor, aspect  )
END;

OBJECTTYPE pmosfetType [
    pmosfetClass : pmosfetType,
      fetClass : fetType,
      TransistorClass : TransistorType,
      analog_componentClass : analog_componentType,
      analog_componentsClass : analog_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF fetType [
    fetClass,
          TransistorClass,
          analog_componentClass,
          analog_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
                Type : STRING;
  IMPLEMENTATION
  METHODS
    method_pmosfetType() READONLY; {};
END;

CLASS pmosfet METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE pmosfetType [
pmosfet,fet,Transistor,analog_component,analog_components,component,
components,pins]
  INIT  pmosfet->defCategory( fet, aspect  )
END;
```

```
OBJECTTYPE nmosfetType [
    nmosfetClass : nmosfetType,
      fetClass : fetType,
      TransistorClass : TransistorType,
      analog_componentClass : analog_componentType,
      analog_componentsClass : analog_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF fetType [
    fetClass,
          TransistorClass,
          analog_componentClass,
          analog_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
                Type : STRING;
  IMPLEMENTATION
  METHODS
    method_nmosfetType() READONLY; {};
END;

CLASS nmosfet METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE nmosfetType [
nmosfet,fet,Transistor,analog_component,analog_components,component,
components,pins]
  INIT  nmosfet->defCategory( fet, aspect  )
END;

OBJECTTYPE pnpType [
    pnpClass : pnpType,
      TransistorClass : TransistorType,
      analog_componentClass : analog_componentType,
      analog_componentsClass : analog_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF TransistorType [
    TransistorClass,
```

```
          analog_componentClass,
          analog_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
                Type : STRING;
  IMPLEMENTATION
  METHODS
    method_pnpType() READONLY; {};
END;

CLASS pnp METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE pnpType [
pnp,Transistor,analog_component,analog_components,component,
components,pins]
  INIT  pnp->defCategory( Transistor, aspect  )
END;

OBJECTTYPE npnType [
    npnClass : npnType,
      TransistorClass : TransistorType,
      analog_componentClass : analog_componentType,
      analog_componentsClass : analog_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF TransistorType [
    TransistorClass,
          analog_componentClass,
          analog_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
                Type : STRING;
  IMPLEMENTATION
  METHODS
    method_npnType() READONLY; {};
END;
```

```
CLASS npn METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE npnType [
npn,Transistor,analog_component,analog_components,component,
components,pins]
  INIT  npn->defCategory( Transistor, aspect  )
END;

OBJECTTYPE resistorType [
    resistorClass : resistorType,
      analog_componentClass : analog_componentType,
      analog_componentsClass : analog_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF analog_componentType [
    analog_componentClass,
          analog_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Value : REAL;
  IMPLEMENTATION
  METHODS
    method_resistorType() READONLY; {};
END;

CLASS resistor  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE resistorType [
resistor,analog_component,analog_components,component,components,pins
]
  INIT  resistor->defCategory( analog_component, aspect  )
END;

OBJECTTYPE capacitorType [
    capacitorClass : capacitorType,
      analog_componentClass : analog_componentType,
      analog_componentsClass : analog_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
```

```
      pinsClass : pinsType ]
        SUBTYPEOF analog_componentType [
    analog_componentClass,
          analog_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Value : REAL;
  IMPLEMENTATION
  METHODS
    method_capacitorType() READONLY; {};
END;

CLASS capacitor METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE capacitorType [
capacitor,analog_component,analog_components,component,components,
pins]
  INIT  capacitor->defCategory( analog_component, aspect  )
END;

OBJECTTYPE power_supplyType [
    power_supplyClass : power_supplyType,
      analog_componentClass : analog_componentType,
      analog_componentsClass : analog_componentsType,
      componentClass : componentType,
      componentsClass : componentsType,
      pinsClass : pinsType ]
        SUBTYPEOF analog_componentType [
    analog_componentClass,
          analog_componentsClass,
          componentClass,
          componentsClass,
          pinsClass ];
  INTERFACE
  PROPERTIES
    Voltage : REAL;
  IMPLEMENTATION
  METHODS
    method_power_supplyType() READONLY; {};
END;
```

```
CLASS power_supply  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE power_supplyType [
power_supply,analog_component,analog_components,component,components,
pins]
  INIT  power_supply->defCategory( analog_component, aspect  )
END;

OBJECTTYPE circuitType [
    circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
    Modality : STRING;
    Name : STRING;
    Label : STRING;
    Position : POINTTYPE;
    Connected_to : connectionsClass;
    Circuit_junctions : junctionsClass;
    Circuit_of : pc_boardClass;
  IMPLEMENTATION
  METHODS
    method_circuitType() READONLY; {};
END;

CLASS circuit
  INSTTYPE circuitType [
    circuit, pc_board, junctions, connections ]
END;

OBJECTTYPE digital_circuitType [
    digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF circuitType [
```

```
    circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Num_output : INT;
    Num_Input : INT;
    Dig_comps : board_digital_componentsClass;
    memberof1 : digital_circuitsClass;
  IMPLEMENTATION
  METHODS
    method_digital_circuitType() READONLY; {};
END;

CLASS digital_circuit METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE digital_circuitType [
digital_circuit,digital_circuits,board_digital_components,circuit,
pc_board,junctions,connections]
  INIT  digital_circuit->defCategory( circuit, aspect  )
END;

OBJECTTYPE digital_circuitsType [
    digital_circuitsClass : digital_circuitsType,
      digital_circuitClass : digital_circuitType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
               Purpose : STRING;
               Cardinality : INT;
               setof : digital_circuitClass;
  IMPLEMENTATION
  METHODS
    method_digital_circuitsType() READONLY; {};
END;

CLASS digital_circuits
  INSTTYPE digital_circuitsType [
    digital_circuits, digital_circuit ]
END;

OBJECTTYPE primitive_seq_circuitType [
```

```
    primitive_seq_circuitClass : primitive_seq_circuitType,
      primitive_seq_circuitsClass : primitive_seq_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF digital_circuitType [
    digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Output : INT;
    Special_name : STRING;
    memberof2 : primitive_seq_circuitsClass;
  IMPLEMENTATION
  METHODS
    method_primitive_seq_circuitType() READONLY; {};
END;

CLASS primitive_seq_circuit METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE primitive_seq_circuitType [
    primitive_seq_circuit, primitive_seq_circuits,
    digital_circuit, digital_circuits,
    board_digital_components, circuit,
    pc_board, junctions, connections ]
  INIT  primitive_seq_circuit->defCategory( digital_circuit, aspect  )
END;

OBJECTTYPE primitive_seq_circuitsType [
    primitive_seq_circuitsClass : primitive_seq_circuitsType,
      primitive_seq_circuitClass : primitive_seq_circuitType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
```

```
                Purpose : STRING;
                Cardinality : INT;
                setof : primitive_seq_circuitClass;
  IMPLEMENTATION
  METHODS
    method_primitive_seq_circuitsType() READONLY; {};
END;

CLASS primitive_seq_circuits
  INSTTYPE primitive_seq_circuitsType [
    primitive_seq_circuits, primitive_seq_circuit ]
END;

OBJECTTYPE LatchType [
    LatchClass : LatchType,
      primitive_seq_circuitClass : primitive_seq_circuitType,
      primitive_seq_circuitsClass : primitive_seq_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF primitive_seq_circuitType [
    primitive_seq_circuitClass,
          primitive_seq_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Con_bit : INT;
  IMPLEMENTATION
  METHODS
    method_LatchType() READONLY; {};
END;
```

```
CLASS Latch METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE LatchType [
Latch,primitive_seq_circuit,primitive_seq_circuits,digital_circuit,
digital_circuits,board_digital_components,circuit,pc_board,junctions,
connections]
  INIT  Latch->defCategory( primitive_seq_circuit, aspect  )
END;

OBJECTTYPE sr_latchType [
    sr_latchClass : sr_latchType,
      LatchClass : LatchType,
      primitive_seq_circuitClass : primitive_seq_circuitType,
      primitive_seq_circuitsClass : primitive_seq_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF LatchType [
    LatchClass,
          primitive_seq_circuitClass,
          primitive_seq_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    R_input : INT;
    S_input : INT;
  IMPLEMENTATION
  METHODS
    method_sr_latchType() READONLY; {};
END;

CLASS sr_latch  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE sr_latchType [
```

```
    sr_latch, Latch, primitive_seq_circuit,
    primitive_seq_circuits, digital_circuit,
    digital_circuits, board_digital_components,
    circuit, pc_board, junctions, connections ]
  INIT  sr_latch->defCategory( Latch, aspect  )
END;

OBJECTTYPE d_latchType [
    d_latchClass : d_latchType,
      LatchClass : LatchType,
      primitive_seq_circuitClass : primitive_seq_circuitType,
      primitive_seq_circuitsClass : primitive_seq_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF LatchType [
    LatchClass,
          primitive_seq_circuitClass,
          primitive_seq_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    D_input : INT;
  IMPLEMENTATION
  METHODS
    method_d_latchType() READONLY; {};
END;

CLASS d_latch METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE d_latchType [
    d_latch, Latch, primitive_seq_circuit,
    primitive_seq_circuits, digital_circuit,
```

```
    digital_circuits, board_digital_components,
    circuit, pc_board, junctions, connections ]
  INIT  d_latch->defCategory( Latch, aspect  )
END;

OBJECTTYPE Flip_flopType [
    Flip_flopClass : Flip_flopType,
      primitive_seq_circuitClass : primitive_seq_circuitType,
      primitive_seq_circuitsClass : primitive_seq_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF primitive_seq_circuitType [
    primitive_seq_circuitClass,
          primitive_seq_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Prev_output : INT;
  IMPLEMENTATION
  METHODS
    method_Flip_flopType() READONLY; {};
END;

CLASS Flip_flop METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE Flip_flopType [
Flip_flop,primitive_seq_circuit,primitive_seq_circuits,
digital_circuit,digital_circuits,board_digital_components,circuit,
pc_board,junctions,connections]
  INIT  Flip_flop->defCategory( primitive_seq_circuit, aspect  )
END;
```

```
OBJECTTYPE d_flip_flopType [
    d_flip_flopClass : d_flip_flopType,
      Flip_flopClass : Flip_flopType,
      primitive_seq_circuitClass : primitive_seq_circuitType,
      primitive_seq_circuitsClass : primitive_seq_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF Flip_flopType [
    Flip_flopClass,
          primitive_seq_circuitClass,
          primitive_seq_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    D_input : INT;
  IMPLEMENTATION
  METHODS
    method_d_flip_flopType() READONLY; {};
END;

CLASS d_flip_flop METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE d_flip_flopType [
    d_flip_flop, Flip_flop, primitive_seq_circuit,
  primitive_seq_circuits, digital_circuit,
  digital_circuits, board_digital_components,
  circuit, pc_board, junctions, connections ]
  INIT  d_flip_flop->defCategory( Flip_flop, aspect  )
END;

OBJECTTYPE jk_flip_flopType [
    jk_flip_flopClass : jk_flip_flopType,
```

```
      Flip_flopClass : Flip_flopType,
      primitive_seq_circuitClass : primitive_seq_circuitType,
      primitive_seq_circuitsClass : primitive_seq_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
    board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF Flip_flopType [
    Flip_flopClass,
          primitive_seq_circuitClass,
          primitive_seq_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    K_input : INT;
    J_input : INT;
  IMPLEMENTATION
  METHODS
    method_jk_flip_flopType() READONLY; {};
END;

CLASS jk_flip_flop  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE jk_flip_flopType [
    jk_flip_flop, Flip_flop, primitive_seq_circuit,
    primitive_seq_circuits, digital_circuit,
    digital_circuits, board_digital_components,
    circuit, pc_board, junctions, connections ]
  INIT  jk_flip_flop->defCategory( Flip_flop, aspect  )
END;

OBJECTTYPE complex_seq_circuitType [
    complex_seq_circuitClass : complex_seq_circuitType,
      complex_seq_circuitsClass : complex_seq_circuitsType,
```

```
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF digital_circuitType [
    digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Num_flip_flops : INT;
    Special_name : STRING;
    memberof : complex_seq_circuitsClass;
  IMPLEMENTATION
  METHODS
    method_complex_seq_circuitType() READONLY; {};
END;

CLASS complex_seq_circuit METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE complex_seq_circuitType [
complex_seq_circuit,complex_seq_circuits,digital_circuit,
digital_circuits,board_digital_components,circuit,pc_board,junctions,
connections]
  INIT  complex_seq_circuit->defCategory( digital_circuit, aspect  )
END;

OBJECTTYPE complex_seq_circuitsType [
    complex_seq_circuitsClass : complex_seq_circuitsType,
      complex_seq_circuitClass : complex_seq_circuitType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
               Purpose : STRING;
               Cardinality : INT;
               setof : complex_seq_circuitClass;
```

```
  IMPLEMENTATION
  METHODS
    method_complex_seq_circuitsType() READONLY; {};
END;

CLASS complex_seq_circuits
  INSTTYPE complex_seq_circuitsType [
    complex_seq_circuits, complex_seq_circuit ]
END;

OBJECTTYPE counterType [
    counterClass : counterType,
      complex_seq_circuitClass : complex_seq_circuitType,
      complex_seq_circuitsClass : complex_seq_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF complex_seq_circuitType [
    complex_seq_circuitClass,
          complex_seq_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Function : STRING;
    Num_bits : INT;
  IMPLEMENTATION
  METHODS
    method_counterType() READONLY; {};
END;

CLASS counter METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE counterType [
```

```
counter,complex_seq_circuit,complex_seq_circuits,digital_circuit,
digital_circuits,board_digital_components,circuit,pc_board,junctions,
connections]
  INIT  counter->defCategory( complex_seq_circuit, aspect  )
END;

OBJECTTYPE ramType [
    ramClass : ramType,
      complex_seq_circuitClass : complex_seq_circuitType,
      complex_seq_circuitsClass : complex_seq_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF complex_seq_circuitType [
    complex_seq_circuitClass,
          complex_seq_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Function : STRING;
    Address : INT;
    Wordsize : INT;
    Capacity : INT;
    Type : STRING;
  IMPLEMENTATION
  METHODS
    method_ramType() READONLY; {};
END;

CLASS ram METACLASS R_SPEC_C_SPEC_CLASS
  INSTTYPE ramType [
ram,complex_seq_circuit,complex_seq_circuits,digital_circuit,
```

```
digital_circuits,board_digital_components,circuit,pc_board,junctions,
connections]
  INIT  ram->defRoleClass(  board_chip  );
    ram->defCategory( complex_seq_circuit, aspect  )
END;

OBJECTTYPE sramType [
    sramClass : sramType,
      ramClass : ramType,
      complex_seq_circuitClass : complex_seq_circuitType,
      complex_seq_circuitsClass : complex_seq_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF ramType [
    ramClass,
          complex_seq_circuitClass,
          complex_seq_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    No_of_ffs : INT;
  IMPLEMENTATION
  METHODS
    method_sramType() READONLY; {};
END;

CLASS sram  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE sramType [
sram,ram,complex_seq_circuit,complex_seq_circuits,digital_circuit,
digital_circuits,board_digital_components,circuit,pc_board,junctions,
connections]
```

```
  INIT  sram->defCategory( ram, aspect  )
END;

OBJECTTYPE dramType [
    dramClass : dramType,
      ramClass : ramType,
      complex_seq_circuitClass : complex_seq_circuitType,
      complex_seq_circuitsClass : complex_seq_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF ramType [
    ramClass,
          complex_seq_circuitClass,
          complex_seq_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    No_of_caps : INT;
  IMPLEMENTATION
  METHODS
    method_dramType() READONLY; {};
END;

CLASS dram  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE dramType [
dram,ram,complex_seq_circuit,complex_seq_circuits,digital_circuit,
digital_circuits,board_digital_components,circuit,pc_board,junctions,
connections]
  INIT  dram->defCategory( ram, aspect  )
END;
```

```
OBJECTTYPE combinational_circuitType [
    combinational_circuitClass : combinational_circuitType,
      combinational_circuitsClass : combinational_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF digital_circuitType [
    digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Special_name : STRING;
    Purpose : STRING;
    memberof2 : combinational_circuitsClass;
  IMPLEMENTATION
  METHODS
    method_combinational_circuitType() READONLY; {};
END;

CLASS combinational_circuit METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE combinational_circuitType [
combinational_circuit,combinational_circuits,digital_circuit,
digital_circuits,board_digital_components,circuit,pc_board,junctions,
connections]
  INIT  combinational_circuit->defCategory( digital_circuit, aspect  )
END;

OBJECTTYPE combinational_circuitsType [
    combinational_circuitsClass : combinational_circuitsType,
      combinational_circuitClass : combinational_circuitType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
```

```
                Purpose : STRING;
                Cardinality : INT;
                setof : combinational_circuitClass;
  IMPLEMENTATION
  METHODS
    method_combinational_circuitsType() READONLY; {};
END;

CLASS combinational_circuits
  INSTTYPE combinational_circuitsType [
    combinational_circuits, combinational_circuit ]
END;

OBJECTTYPE half_adderType [
    half_adderClass : half_adderType,
      combinational_circuitClass : combinational_circuitType,
      combinational_circuitsClass : combinational_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF combinational_circuitType [
    combinational_circuitClass,
          combinational_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Function : STRING;
  IMPLEMENTATION
  METHODS
    method_half_adderType() READONLY; {};
END;
```

```
CLASS half_adder  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE half_adderType [
    half_adder, combinational_circuit,
    combinational_circuits, digital_circuit,
    digital_circuits, board_digital_components,
    circuit, pc_board, junctions, connections ]
  INIT  half_adder->defCategory( combinational_circuit, aspect  )
END;

OBJECTTYPE full_adderType [
    full_adderClass : full_adderType,
      combinational_circuitClass : combinational_circuitType,
      combinational_circuitsClass : combinational_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF combinational_circuitType [
    combinational_circuitClass,
          combinational_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Function : STRING;
  IMPLEMENTATION
  METHODS
    method_full_adderType() READONLY; {};
END;

CLASS full_adder  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE full_adderType [
    full_adder, combinational_circuit,
    combinational_circuits, digital_circuit,
```

```
    digital_circuits, board_digital_components,
    circuit, pc_board, junctions, connections ]
  INIT  full_adder->defCategory( combinational_circuit, aspect  )
END;

OBJECTTYPE decoderType [
    decoderClass : decoderType,
      combinational_circuitClass : combinational_circuitType,
      combinational_circuitsClass : combinational_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF combinational_circuitType [
    combinational_circuitClass,
          combinational_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Num_bits : INT;
  IMPLEMENTATION
  METHODS
    method_decoderType() READONLY; {};
END;

CLASS decoder METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE decoderType [
    decoder, combinational_circuit,
    combinational_circuits, digital_circuit,
    digital_circuits, board_digital_components,
    circuit, pc_board, junctions, connections ]
  INIT  decoder->defCategory( combinational_circuit, aspect  )
END;
```

```
OBJECTTYPE encoderType [
    encoderClass : encoderType,
      combinational_circuitClass : combinational_circuitType,
      combinational_circuitsClass : combinational_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF combinational_circuitType [
    combinational_circuitClass,
          combinational_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Num_inputs : INT;
    Num_outputs : INT;
  IMPLEMENTATION
  METHODS
    method_encoderType() READONLY; {};
END;

CLASS encoder METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE encoderType [
    encoder, combinational_circuit,
    combinational_circuits, digital_circuit,
    digital_circuits, board_digital_components,
    circuit, pc_board, junctions, connections ]
  INIT  encoder->defCategory( combinational_circuit, aspect  )
END;

OBJECTTYPE romType [
    romClass : romType,
```

```
      combinational_circuitClass : combinational_circuitType,
      combinational_circuitsClass : combinational_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF combinational_circuitType [
    combinational_circuitClass,
          combinational_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Capacity : INT;
    Wordsize : INT;
    Address : INT;
    Function : STRING;
  IMPLEMENTATION
  METHODS
    method_romType() READONLY; {};
END;

CLASS rom METACLASS R_SPEC_C_SPEC_CLASS
  INSTTYPE romType [
    rom, combinational_circuit,
    combinational_circuits, digital_circuit,
    digital_circuits, board_digital_components,
    circuit, pc_board, junctions, connections ]
  INIT  rom->defRoleClass(  board_chip  );
    rom->defCategory( combinational_circuit, aspect  )
END;

OBJECTTYPE promType [
    promClass : promType,
```

```
      romClass : romType,
      combinational_circuitClass : combinational_circuitType,
      combinational_circuitsClass : combinational_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF romType [
    romClass,
          combinational_circuitClass,
          combinational_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
       INTERFACE
  PROPERTIES
    Type : STRING;
  IMPLEMENTATION
  METHODS
    method_promType() READONLY; {};
END;

CLASS prom  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE promType [
    prom, rom, combinational_circuit,
    combinational_circuits, digital_circuit,
    digital_circuits, board_digital_components,
    circuit, pc_board, junctions, connections ]
  INIT  prom->defCategory( rom, aspect  )
END;

OBJECTTYPE eepromType [
    eepromClass : eepromType,
      promClass : promType,
      romClass : romType,
```

```
      combinational_circuitClass : combinational_circuitType,
      combinational_circuitsClass : combinational_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF promType [
    promClass,
          romClass,
          combinational_circuitClass,
          combinational_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
       INTERFACE
  PROPERTIES
    Spec_type : STRING;
  IMPLEMENTATION
  METHODS
    method_eepromType() READONLY; {};
END;

CLASS eeprom  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE eepromType [
    eeprom, prom, rom, combinational_circuit,
    combinational_circuits, digital_circuit,
    digital_circuits, board_digital_components,
    circuit, pc_board, junctions, connections ]
  INIT  eeprom->defCategory( prom, aspect  )
END;

OBJECTTYPE epromType [
    epromClass : epromType,
      promClass : promType,
      romClass : romType,
```

```
      combinational_circuitClass : combinational_circuitType,
      combinational_circuitsClass : combinational_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF promType [
    promClass,
          romClass,
          combinational_circuitClass,
          combinational_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
       INTERFACE
  PROPERTIES
    Spec_type : STRING;
  IMPLEMENTATION
  METHODS
    method_epromType() READONLY; {};
END;

CLASS eprom METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE epromType [
    eprom, prom, rom, combinational_circuit,
    combinational_circuits, digital_circuit,
    digital_circuits, board_digital_components,
    circuit, pc_board, junctions, connections ]
  INIT  eprom->defCategory( prom, aspect  )
END;

OBJECTTYPE muxType [
    muxClass : muxType,
      combinational_circuitClass : combinational_circuitType,
      combinational_circuitsClass : combinational_circuitsType,
```

```
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF combinational_circuitType [
    combinational_circuitClass,
          combinational_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Num_con_bits : INT;
    Num_inputs : INT;
  IMPLEMENTATION
  METHODS
    method_muxType() READONLY; {};
END;

CLASS mux METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE muxType [
    mux, combinational_circuit, combinational_circuits,
    digital_circuit, digital_circuits,
    board_digital_components, circuit, pc_board,
    junctions, connections ]
  INIT  mux->defCategory( combinational_circuit, aspect  )
END;

OBJECTTYPE demuxType [
    demuxClass : demuxType,
      combinational_circuitClass : combinational_circuitType,
      combinational_circuitsClass : combinational_circuitsType,
      digital_circuitClass : digital_circuitType,
      digital_circuitsClass : digital_circuitsType,
      board_digital_componentsClass : board_digital_componentsType,
```

```
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF combinational_circuitType [
    combinational_circuitClass,
          combinational_circuitsClass,
          digital_circuitClass,
          digital_circuitsClass,
          board_digital_componentsClass,
          circuitClass,
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Num_outputs : INT;
    Num_con_bits : INT;
  IMPLEMENTATION
  METHODS
    method_demuxType() READONLY; {};
END;

CLASS demux METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE demuxType [
    demux, combinational_circuit, combinational_circuits,
    digital_circuit, digital_circuits,
    board_digital_components, circuit, pc_board,
    junctions, connections ]
  INIT  demux->defCategory( combinational_circuit, aspect  )
END;

OBJECTTYPE analog_circuitType [
    analog_circuitClass : analog_circuitType,
      analog_circuitsClass : analog_circuitsType,
      board_analog_componentsClass : board_analog_componentsType,
      circuitClass : circuitType,
      pc_boardClass : pc_boardType,
      junctionsClass : junctionsType,
      connectionsClass : connectionsType ]
        SUBTYPEOF circuitType [
    circuitClass,
```

```
          pc_boardClass,
          junctionsClass,
          connectionsClass ];
  INTERFACE
  PROPERTIES
    Analog_comps : board_analog_componentsClass;
    memberof1 : analog_circuitsClass;
  IMPLEMENTATION
  METHODS
    method_analog_circuitType() READONLY; {};
END;

CLASS analog_circuit  METACLASS CATEGORY_SPECIALIZATION_CLASS
  INSTTYPE analog_circuitType [
    analog_circuit, analog_circuits, board_analog_components,
    circuit, pc_board, junctions, connections ]
  INIT  analog_circuit->defCategory( circuit, aspect  )
END;

OBJECTTYPE analog_circuitsType [
    analog_circuitsClass : analog_circuitsType,
      analog_circuitClass : analog_circuitType ]
        SUBTYPEOF Metaclass_InstType;
  INTERFACE
  PROPERTIES
               Purpose : STRING;
               Cardinality : INT;
               setof : analog_circuitClass;
  IMPLEMENTATION
  METHODS
    method_analog_circuitsType() READONLY; {};
END;

CLASS analog_circuits
  INSTTYPE analog_circuitsType [
    analog_circuits, analog_circuit ]
END;

END_SCHEMA;
```

APPENDIX C

DIAGRAMS

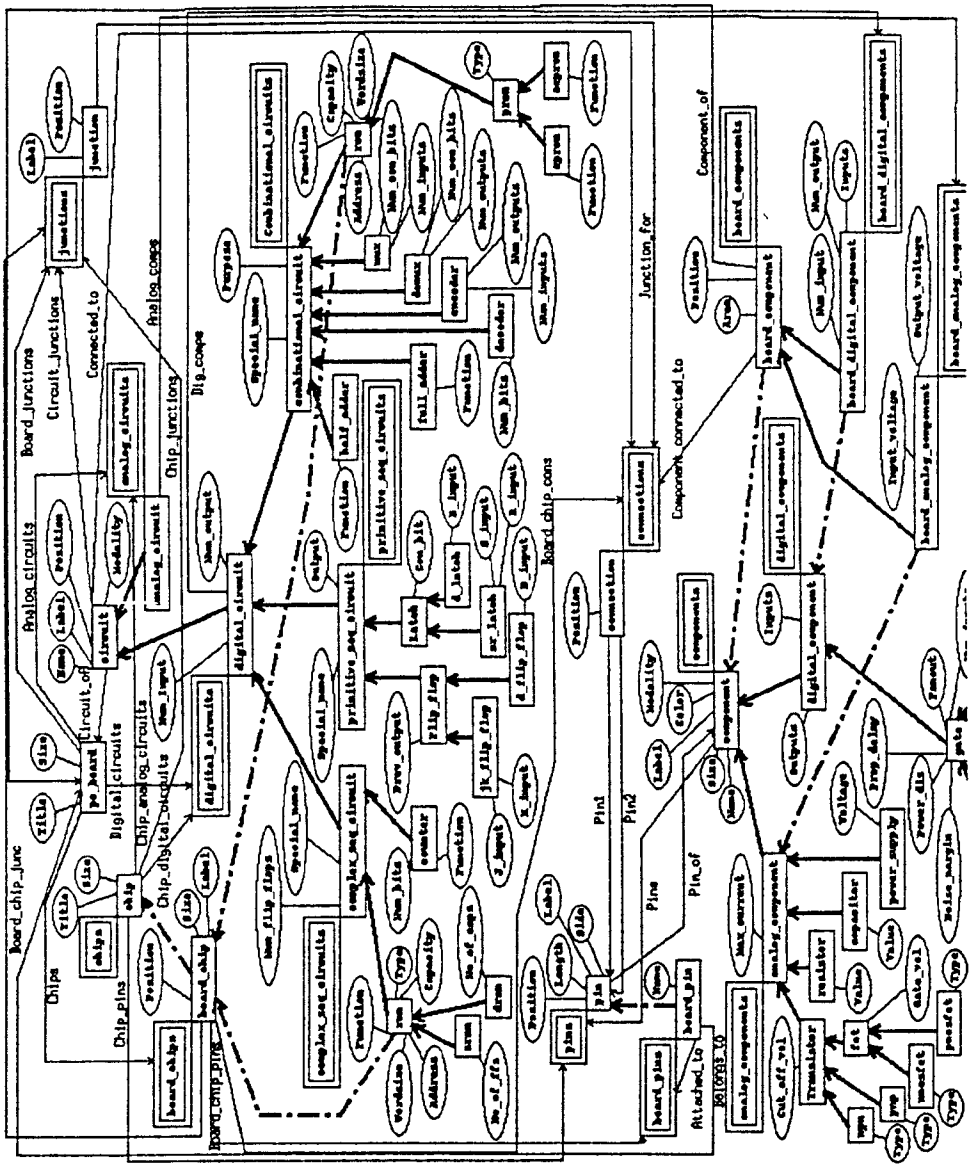

Figure C.1: Whole Schema 1

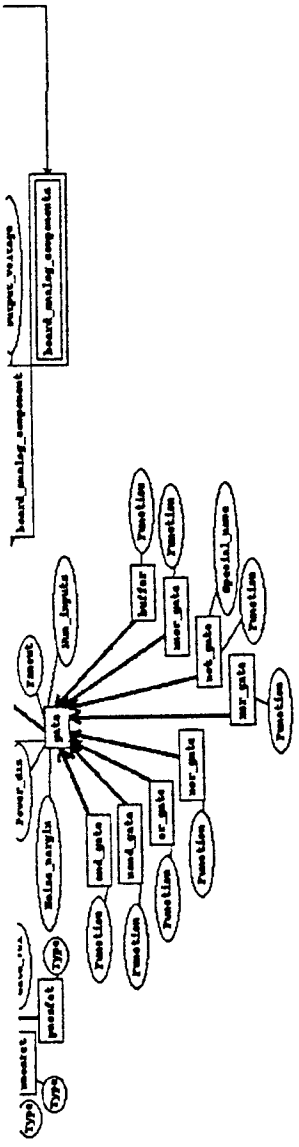

Figure C.2: Whole Schema 2

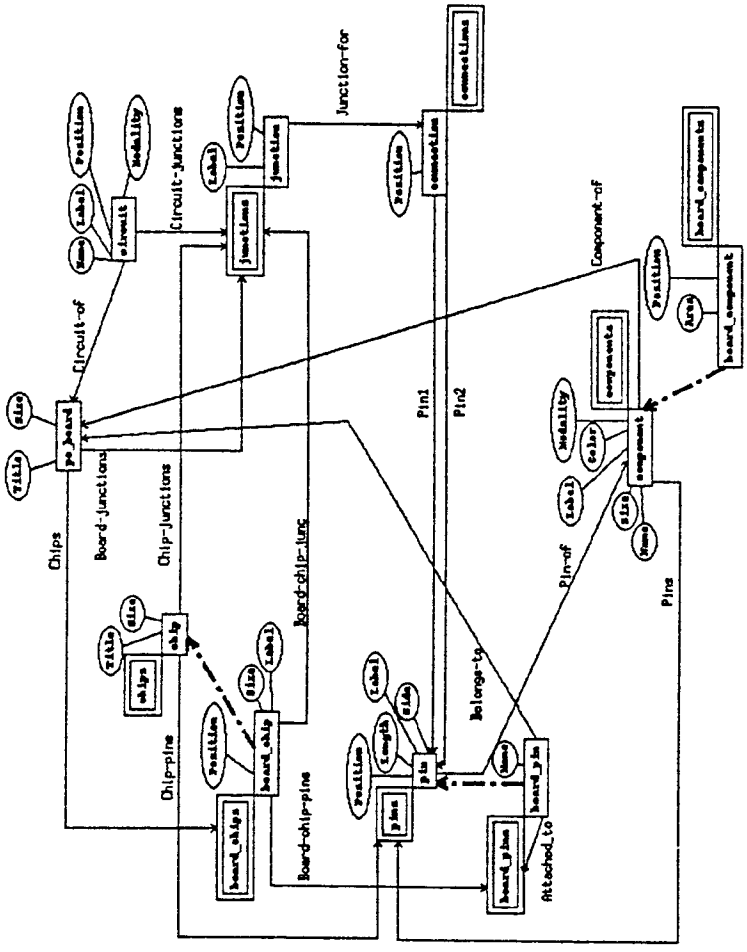

Figure C.3: Part 1 of the Whole Schema

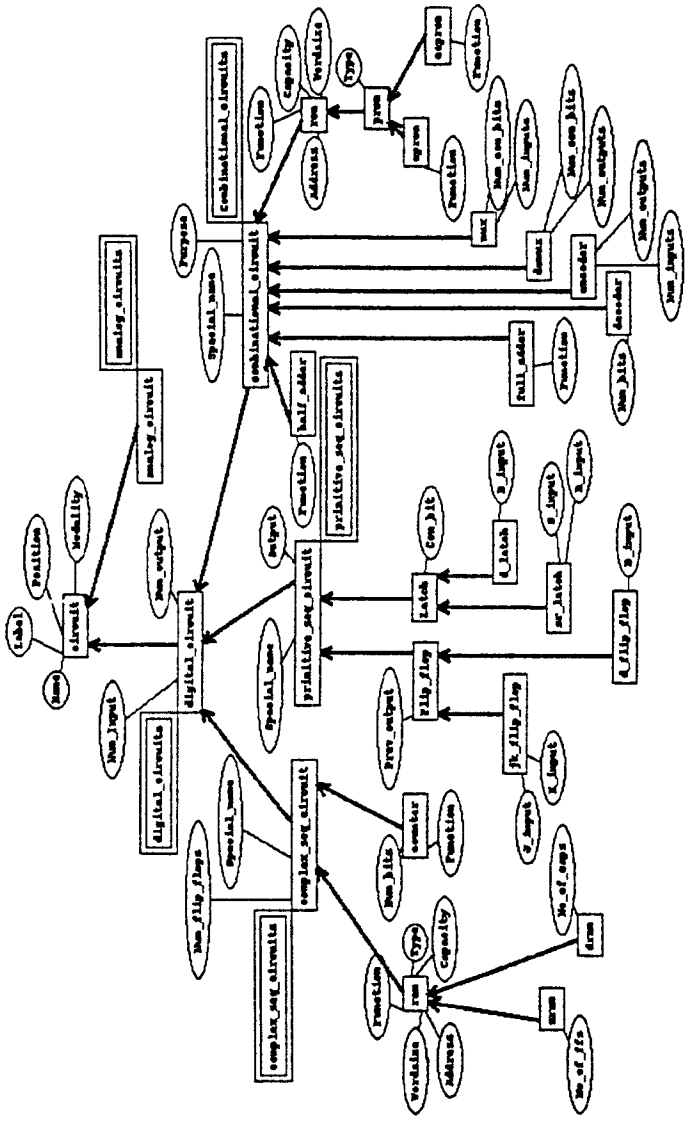

Figure C.4: Part 2 of the Whole Schema

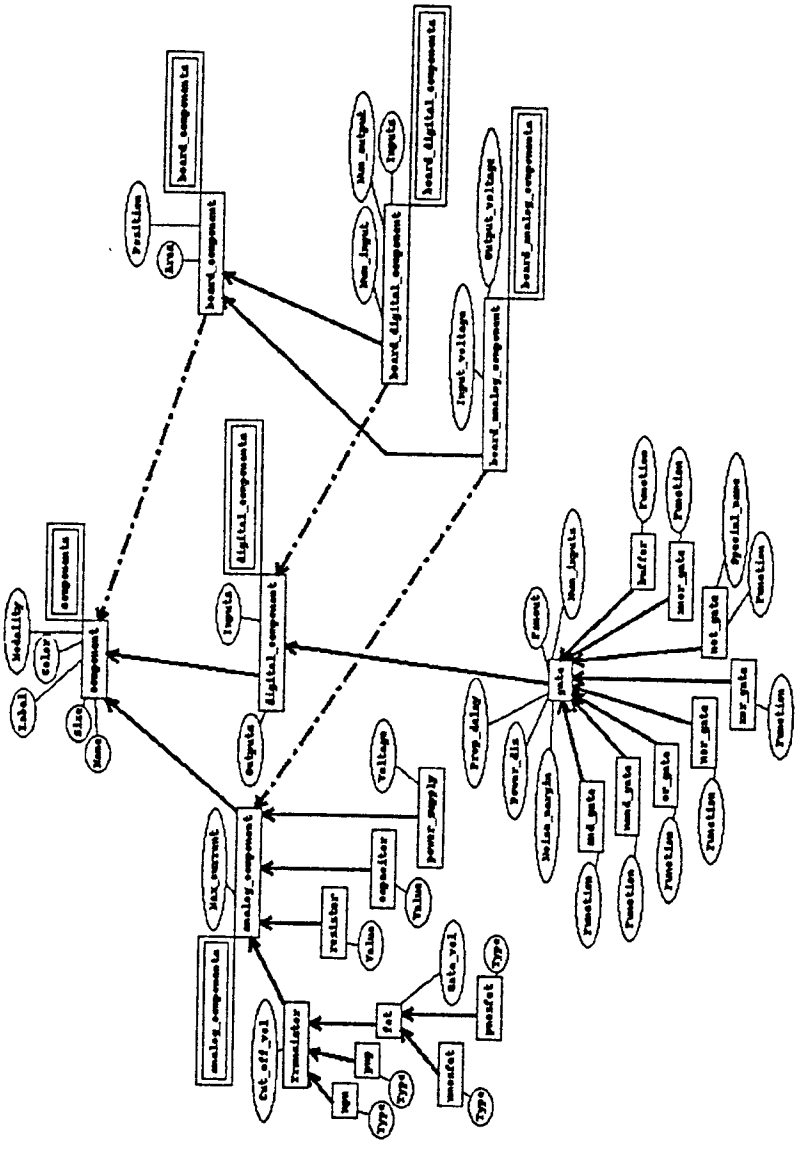

Figure C.5: Part 3 of the Whole Schema

REFERENCES

[1] James Geller, Yehoshua Perl. "Separating Structural and Semantic Elements in Object Oriented Knowledge Base" *Proceedings of Advanced Database System Symposium'89, Kyoto, Japan, December 7-8, 1989*

[2] W. Klas, E.J. Neuhold, etc. "VML-The VODAK Data Modeling Language" Technical Report, *Integrated Publication and Information System Institute, GMD, 1991*

[3] J. Geller. "A Knowledge Representation Theory for Natural Language Graphics" *PhD Dissertation, appeared as Tech Report 88-15, CS Department, SUNY at Buffalo, 1988*

[4] J. Geller. "Propositional Representation for Graphical Knowledge" *Int. Journal Man-Machine Studies', 34, 1991, PP. 97-131*

[5] J. Geller and S. C. Shapiro. "Graphical Deep Knowledge for Intelligent Machine Drafting" *Tenth International Joint Conference on AI, Morgan Kaufman Publishers Inc., Los Altos, CA, 1987, pp. 545-551*

[6] Prasanna S VenKatesh. "Representation of Graphical Deep Knowledge in an Object-Oriented Database System" *Master Thesis, appeared in the Proceedings of Objected-Oriented Database Management, San Francisco, 1991*

[7] James Geller, Yehoshua Perl, Erich Neuhold. "Structrual Schema Integration in Heterogeneous Multi-Database Systems Using the Dual Model" *Proceedings of Interoperability in Multidatabase Systems, IMS 91, Kyoto, Japan, April 7-9, 1991*

[8] A.Kemper and M.Wallrath. "An Analysis of Geometric Modeling in Database Systems", *ACM Computing Surveys, 1987, pp. 47-91.*

[9] R. Lorie and W. Plouffe."Complex Objects and Their Use in Design Transactions" *Proceedings of ACM SIGMOD Conference on Engineering Design Applications, San Jose, California, May 1983, pp. 115-121.*

[10] E. Batory and W. Kim. "Modeling Concepts for VLSI CAD Objects" *ACM Transactions on Database Systems, 1985, pp. 322-346.*

[11] S. C. Shapiro and James Geller. "Artificial Intelligence and Automated Design" *Principles of Computer Aided Design: Computability of Design, Wiley Interscience, 1987, pp. 173-187*

[12] S. C. Shapiro. "The SNePS Semantic Network Processing System" *Associative Networks: The Representation and Use of Knowledge by Computer, N. V. Findler(editor), Academic Press, New York, 1979.*

[13] Tomihisa Kamada and Satoru Kawai. "A General Framework for Visualizing Abstract Objects and Relations." *ACM Trans. Graphics, Vol. 10, No. 1, Jan. 1991, Pages 1-39.*

[14] A. Borning and R. Duisberg. "Constrain-based Tools for Building User Interfaces" *ACM Trans. Graph. 5, 4 (Oct. 1986), 345-374.*

[15] Y. Rens, L. Miller S.C. Shapiro and N.K. Sondheimer. "Automatic Construction of User-Interface Displays" *Proceedings of AAAI-88(Minnesota, Aug. 1988). 808-813.*

[16] S. Khoshafian, R. Abnous. *Object-Oriented Concepts, Languages, Database and User Interfaces* John Wiley & Sons Inc., 1990

[17] B. Stroustrup. *The C++ Programming Language* Second Edition, Addison-Wesley Publishing Company, 1991

[18] Douglas A. Young. *Object-Oriented Programming with C++ and OSF/Motif* Prentice-Hall, Inc., 1992