LARGE DATABASE ADA PROGRAM FOR REAL TIME LABORATORY INSTRUMENT CONTROL AND DATA ACQUISITION

RICHARD G. SARTORE
ARMY RESEARCH LABORATORY
AMSRL-EP-RA
FORT MONMOUTH, NJ 07703-5601
908-544-2261, FAX-908-532-0156
RSARTORE@MONMOUTH-ETDL1.ARMY.MIL

1. INTRODUCTION.

An ADA program, called CADET (Computer Aided Diagnostic E-Beam Testing) was developed under Contract DAAL01-85-0440 (Ref. 1). CADET integrates the retrieval and display of VLSI/VHSIC design information, controls instrumentation for waveform measurement/acquisition, and analyzes measured and simulated data for design verification and failure analysis. The device design information includes the design layout, simulation output, and test pattern files. Typically, these design files can each be 1Mbyte or more. These large data files are translated from the available CAD file formats for organization in a hierarchy of objects and subsequently for quick retrieval and display in CADET.

After delivery to Army Research Laboratory (ARL), the ADA code was modified for use with ARL instrumentations. Specifically, the instrumentation packages for stimuli generator was re-coded for the word generator available at ARL and a new package was added to incorporate motorized stage control in the CADET program. New code development also added a prototype localized simulation capability and the ability to import SPICE simulation files. The unique localized simulator performs automatic model construction from the layout data. The original and new code was extensively tested in the ARL laboratory environment, with many changes made to the ADA code to address operational requirements and to fix detected errors.

Due to the large size of the CAD layout file, a SCAN program was written to pre-scan the content of layout file to extract a range of numbers and to set a scale for efficient transfer of graphics information to the CADET format. Other utilities were written for test pattern data and measured data files.

The CADET program has been used in an operational system for measurements of nodal waveforms and evaluation of new designs at the Army Research Laboratory (Ref. 2-4). CADET was and still is a research software development in ADA that seeks to integrate the many large design and test databases and automated measurements in a transparent manner to the diagnostician. A number of critical issues in the design and test areas have been successfully addressed and implemented in ADA, by providing a

ACM 0-89791-666-2/94/0011--0406

single platform for display and analysis of the various design, test and measurement databases. Further, several unique data structures and control methodologies have been developed in ADA during this software research project.

Although preliminary work and coding has been generated in the areas of image acquisition/analysis, increased automation capability, automated test program generation, fault tracing and measured data analysis, further effort is still required for integration, development and implementation. Specifically, a pattern recognition algorithm needs to be developed/implemented to provide fine positioning of the motorized stage(Ref.5).

The ADA code modularity and maintainability will provide an easier migration path in the future to improved platforms and instrumentations. Because of the ADA modularity, many of the ADA packages can and have been reused for related applications in our laboratory.

2. SYSTEM OVERVIEW

The CADET ADA program was generated on DEC MicroVAX Station II under the VMS operating system, using the DEC certified ADA compiler. ADA was chosen as the language of choice to provide portability and maintainability. In fact, limited portability was established by porting needed sections of CADET code to a PC ADA compiler (Meridian's Open ADA) to develop the SCAN utility for large layout design files. Maintenance of the code was demonstrated by the test and debug work performed at ARL.

The initial test and debug of the ADA code was performed over the NETWORK using the VMS editor. This configuration was workable but made editing of large package unwieldy due to small window size and random slowdowns caused by network traffic. The final solution was to obtain a VMS Language sensitive editor (LSED) and a dedicated ADA compiler for our workstation. This allowed multiple windows for code comparison and editing, which greatly speeded up code maintenance and compilation.

In the interest of portability and maintainability, the graphics interface was established through the standard GKS (Graphics Kernel System) with an ADA binding, under the DEC VAX version. Since CADET is a graphics intensive program, interacting with the diagnostician via menu and use of the mouse, it was essential to provide a standard graphical interface for portability of the code. All graphic related code has been localized in the GRAPHICS package, allowing easy access for GKS upgrades or a convenient means to provide access to other graphics standards, such as, PHIGS.

Communications with the measurement instrumentation was accomplished through the IEEE-STD-488 interface bus, using the VMS IEX IEEE-488 device driver. Since the IEEE-488 bus is an industry standard and available for practically all computers and instrumentations, the portability of the instrumentation control is greatly enhanced. As in the case of the GRAPHICS package, the IEEE-488 related code is localized in a separate ADA package, allowing easy maintenance of the CADET code for IEEE-488 interface bus or adaption to other standards, such as, VXI. Communications with other computers and processes is through the ETHERNET using DECnet.

A generalized view of the CADET hardware and software components is shown in Figure 1. CADET accomplishes all operations through tasks. A task is spawned in each package to handle transactions and messages. The graphical interface with the diagnostician, circuit simulation, measurements, package menus, etc. are implemented through tasks. Communications between tasks is accomplished by passing messages known as directives. Directives are " implemented as ADA variant records with discriminants"(Ref. 1). The specific implementation of the message record is shown in Figure 2.

The message is passed from main program to the instrument controller through the use of the VAX/VMS AST (Asynchronous System Traps) that are handled in ADA as "pragma AST_ENTRY". An example of the implementation in CADET is shown in Figure 3. The main CADET program passes the Message_Record through a communications handling package (Com_Handling) to the Instrument_Controller. The Instrument_Controller handles the Message_Record in the Instrument_Server package, which after routing the message to the appropriate package for action can also pass measured data back through Com_Handling for display in the main CADET program. For communications between computers and processes, a STATUS_SET enumeral of type SEMAPHORE is used to insure that proper buffer size is allocated for long messages, such as, digitized images. This SEMAPHORE construct allows separate programs to be run on separate computers, over the ETHERNET. Since the communications between the two processes is confined to individual ADA packages in each process, any changes or adaptations that have to be made for other platforms can be easily addressed.

3. DATA STRUCTURES

All data structures in CADET were designed from an object oriented standpoint for more compact and structured design of the ADA code. Specifically, the data structures in the CADET program were carefully designed to organize the layout design database file, to capture and maintain the continuity of conductors across modules . The layout file is typically the largest file received from the CAD group. This file contains information on spatial layout of the various conductor layers, of vias and contacts, of diffusions, polysilicon, resistors, capacitors, etc. This is the file that the foundry uses for device fabrication and is critical for CADET, since it provides navigational aids for positioning of the electron beam and moving the motorized stage and providing the ability to trace electrical continuity for signal tracing/back-tracing. The design file used by CADET is in CIF format (Caltech Intermediate Format), which is ASCII based. The CIF layout file provides for definition of circuit modules, with multiple calls for translation, rotation, etc. and placement of the module on the chip surface. Each module contains detailed spatial information on conductors, resistors, etc. As received, the CIF file is flat, that is, all information is displayed at once. CADET takes the CIF file and organizes the data into various objects and cluster objects with their appropriate relationships, to establish hierarchy. The relationships can be binary for connection from one conductor type to another, such as, a via or from one port to another, such as, across module boundaries. The relationship can be trinary to represent transistors. Finally, generalizing, the relationships can be N-ary to represent complex links between the CADET abstract data objects. The basic conceptualization and utilizations for N-ary relationships has been studied and developed in graph and hypergraph theory.

The ADA implementation for an CADET object is " a variant record with discriminant, and at least three associates lists"(Ref. 1). These objects are defined in the package specification HIERARCHY. A separate package TECHNOLOGY assigns graphical display attributes and circuit parameters to the abstract data types in a user defined technology file, to establish technology independence. The ADA definition of the objects and an example of their records is shown in Figure 4. This is a basic CONDUCTOR object TYPE with an extents list (defines spatial details of conductor), a relations list (define relations to other conductors) and link to a higher level CLUSTER object. The CLUSTER object is linked to a higher level MODULE object, which can enter into relationships with other modules, until the complete device design is contained under the ROOT module object.(Figure 5)

An example of the ADA implementation for the CADET RELATIONS object is shown in Figure 6. This object is used to maintain relationships among objects, with each object on multiple lists or N-ary relationships. Modules can be assembled into a hierarchy using the contained relationship. The Placement_Record contains the information used to place the contained module (B) inside the containing module (A). The CONTROL relation defines the transistor object type, which defines the transistor channel by obtaining the union of base sets for a transistor defined in the technology file.

4. IMPORTING EXTERNAL DATABASES

The CIF layout file is preprocessed to extract the required hierarchy and relations information and to store this information into a set of data files that contain the organized data. This facility provides control of the layer of hierarchy details that are displayed when CADET loads the design data, i.e., only information needed at the time of diagnosis needs to be loaded and exposed. The control of information loaded and displayed provides faster data retrieval and draws and ,as a consequence, more productive diagnostic sessions.

In addition, stimuli and functional simulation results must also be preprocessed and saved in CADET file format. As an extension to the original CADET program, a facility has been incorporated into CADET that loads and displays the simulation results in a SPICE file. SPICE is a circuit simulation program using a variety of built-in transistor models and produces ASCII files with detailed timing information of waveforms at specified circuit nodes. The implementation developed at ARL requires that the SPICE output files be edited with a standard text editor to insert one line of information for file parser. Example of the ADA code for Parse_SPICE_File is given in Figure 7. This procedure accepts a file name from prompt and then reads edited SPICE file, loading node names and time-voltage data for each signal, which in turn are selectable in one of the CADET windows

5. CONCLUSIONS

The software effort to develop the ADA program called CADET has been implemented at ARL, with an operational software program for diagnostics of complex microelectronic devices. CADET has over 50K SLOCs and over 40 different packages. Due to the modularity and structured program techniques used in its development, the CADET software is adaptable across platforms, instrumentation technology and device technology.

With the advent of ADA 9x, it has been proposed that the CADET program would be a good testbed to establish the compatibility and migratability of ADA83 code developed on a certified compiler to a 9X certified compiler. A re-write of the CADET objects into the 9X explicit object facility should make CADET code simpler and more maintainable/portable (Ref.6). Of immediate benefit to CADET would be the INHERITANCE facility in 9X.

Due to the large microelectronic designs (VLSI and MCM) being developed, the present hardware configuration (1 MIP) makes layout display very time consuming. As a consequence, it has been proposed that the CADET code be ported to RISC work stations to make use of their graphic acceleration and increased computational capability.

6. ACKNOWLEDGEMENTS

J.Luisi, in addition to developing the original CADET program, has spent several summers developing the prototype localized simulation capability and the motorized stage package. His efforts were critical in developing present CADET ADA code and in-house capabilities. C. Hogh adapted and wrote the ADA packages for the ARL word generator. C.Marshall developed the SPICE import facility and provided much needed expertise in ADA code maintenance/support.

7. REFERENCES

1. S.Hoelke and J.Luisi, Electron-Beam Diagnostics for Microelectronic Design Validation, R&D Technical Report SLCET-TR-85-0440-F, Jan. 1989

2. R.Sartore and M.Royals,"Use of Computer Aided Diagnostic E-Beam Testing (CADET) System to Perform Failure Diagnostics and Design Verification on VLSI Devices", Test Engineering Conference June 24-27, 1991, p53-73

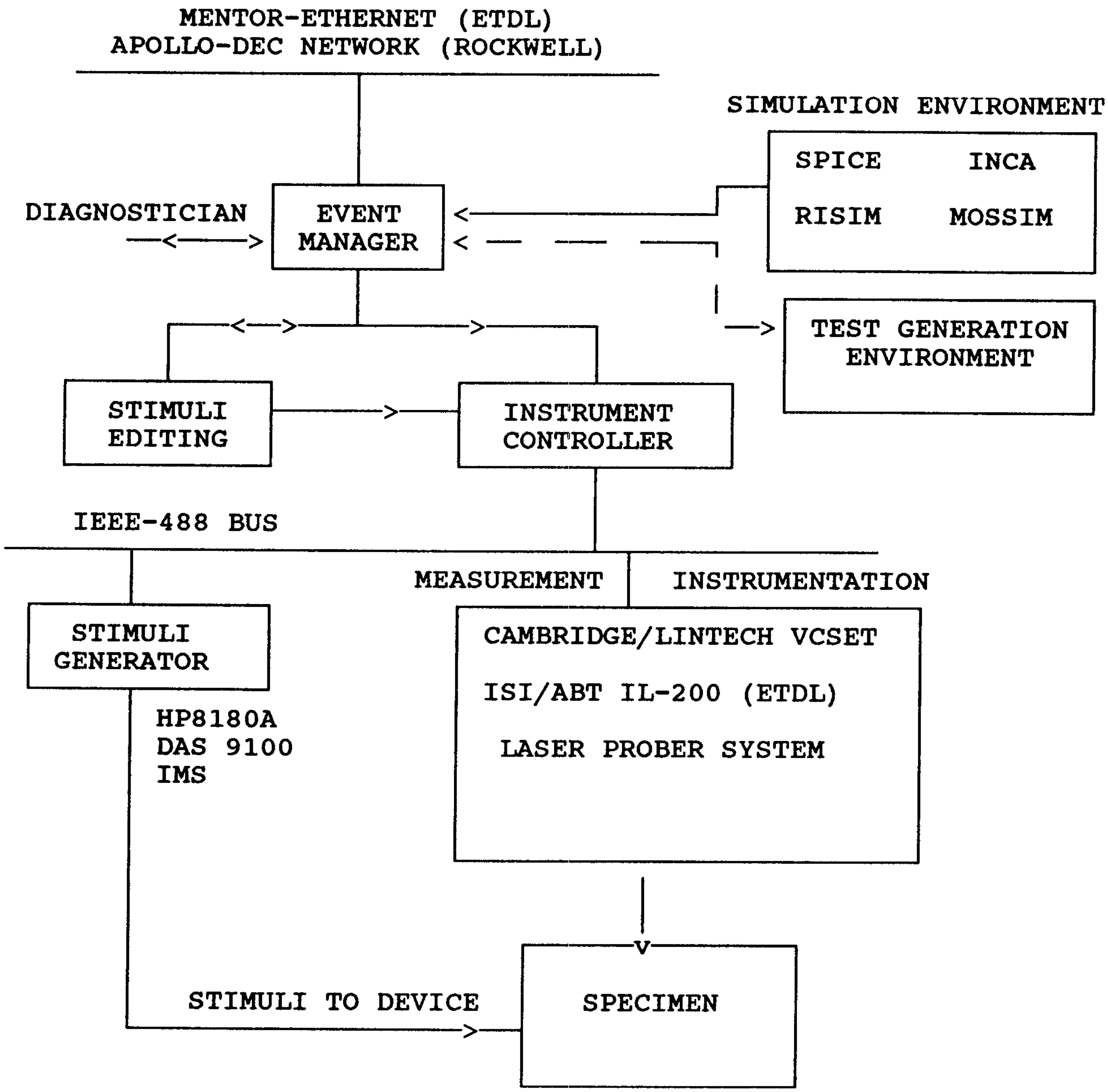
3. R.Sartore and R.Buchanan, Computer Aided E-Beam Testing of Microelectronic Devices, Tutorial Notes for 1991 International Reliability Physics Symposium

4. R.Sartore, J.Erickson and R.Heuner, "Diagnosis of Synthesis/Emulation Problems on a SINCGARS Radio ASIC Using Computer Aided Diagnostic E-Beam Test (CADET) System, GOMAC 1992, P163-164

5. J.Michener,"High Precision Electron Beam Positioning Using Computer Image Analysis for Electron Beam Testing", Microelectronic Engineering 7, (1987),p223-229

6. J.Barnes, "Introducing ADA 9x", ACM ADA Letters, Vol.XII, No.6, Nov/Dec93

CADET CONFIGURATION



Figur 1. CADET Configuration

```
-------------------------------------------------------------------
--      Message Object Types
-------------------------------------------------------------------
   type STATUS_SET is (GET_IMAGE, GET_SITE, GET_WAVE, -- Request.
               GOT_IMAGE, GOT_SITE, GOT_WAVE,-- Response.
               NO_IMAGE,  NO_SITE,  NO_WAVE,  -- Abort or cancel.
               SEMAPHORE, ...., STIMULI);
   type ORIGINATOR_SET is (COMMUNICATOR, ... , STIMULOR);
   type STATE_SET is (CALIBRATED, INITIALIZED, ...., SETUP);

type SITE_OPTION_SET is (TO_REFERENCE_SITE,-- stage registration.
                         TO_NODE_SITE); -- For noise subtraction.

type MESSAGE_RECORD (   OPTION       : STATUS_SET ;
                               ARRAY_SIZE  : natural ) is
record
  TASK_ID   : integer ;
  ORIGINATOR: ORIGINATOR_SET ;
  case OPTION is
    when SEMAPHORE => STATUS : STATUS_SET ;-- Common
                                       -- aprioi signal message.
                      SIZE   : natural ; -- Needed for
                                    -- dynamic record allocation.
    when GET_IMAGE => CENTER_X, CENTER_Y    : float ; -- Microns.
                          HALF_SIZE       : float ;  -- Microns.
                          SAMPLE_TIME     : Bit_Index_Range ;
    when GOT_IMAGE => IMAGE_ROW: IMAGE_ROW_TYPE ; -- send a row
at a time
    when GET_SITE  => SITE_OPTION : SITE_OPTION_SET ;
    when GOT_SITE  => SITE_XX, SITE_YY       : float ;  -- Microns.
    when GET_WAVE  => SITE_X, SITE_Y         : float ;  -- Microns.
                      START_TIME, STOP_TIME : Bit_Index_Range ;
                      TIME_RESOLUTION       : TIME_TYPE ;
                      VOLTAGE_RESOLUTION    : VOLTAGE_TYPE ;
    when GOT_WAVE  => WAVE          : WAVE_TYPE(1..ARRAY_SIZE) ;

      ---------------- additional test conditions
  end case ;
end record ;

type MESSAGE_POINTER is access MESSAGE_RECORD ;
```

Figure 2. ADA Data Structure Used for Message Passing

```
 -- declare variables

 task PROBER_Com_SERVER is
   entry Send( Message : Data_Types.Message_Pointer );
   entry RECEIVE_AST ;
   pragma ast_entry(RECEIVE_AST) ;
 end PROBER_Com_SERVER ;

-------------------------------------------------------

task body Prober_Com_SERVER is

 -- declare variables

begin--Prober_Com_Server
   accept Send( Message : Data_Types.Message_Pointer ) do
      Message_To_Be_Sent := Message;
   end Send;
   case Message_To_Be_Sent.Option is

      when   --- check for shutdown
             --- otherwise proceed to following code

        Arm_Prober_AST;
      loop
         select
            accept RECEIVE_AST do
               Text_IO.Put_Line( "AST triggered in PROBER_AST_SERVER"

               --- service AST

            end Receive_AST;

               --- get data, store file or put on a stack

             Arm_Prober_AST;

         or
            accept Send( Message : Data_Types.Message_Pointer ) do
               Message_To_Be_Sent := Message;
            end Send;

               --- send message

         end select;
      end loop ;
   end case;
end PROBER_Com_SERVER ;
```

Figure 3. Example Use of AST_ENTRY in COM_HANDLING

```
--Objects
   type Object_Types is ( Conductor, Zone, Cluster, Module,
                          Subnode, Node, Model );
   type Object_Record( Object_Type : Object_Types );
   type Pointer_To_Object_Record is access Object_Record;

--Relations
   type Relation_Types is ( Port, Contained, Connected,
          Conducted, Control, Coupled, I_Coupled, Stimuli );
   type Relation_Record( Relation_Type : Relation_Types );
   type Pointer_To_Relation_Record is access Relation_Record;

   type Object_Record( Object_Type : Object_Types ) is
    record
     Extents   : Pointer_To_Extent_Record   := null;
     Relations : Pointer_To_Relation_Record := null;
     Link      : Pointer_To_Object_Record   := null;

     Index     : Short_Natural; -- For IO & Crossreferencing
                                -- to module array
     Rel_Loc   : File_Location_Record;--required to span
                                      -- delayed loading
     case Object_Type is
        when Conductor => Conductor_Type :
                          Technology.Class_Member_Indices;
        when others    => Details : Pointer_To_Details_Record;
     end case;
  end record;
```

Figure 4. Example of CADET OBJECT

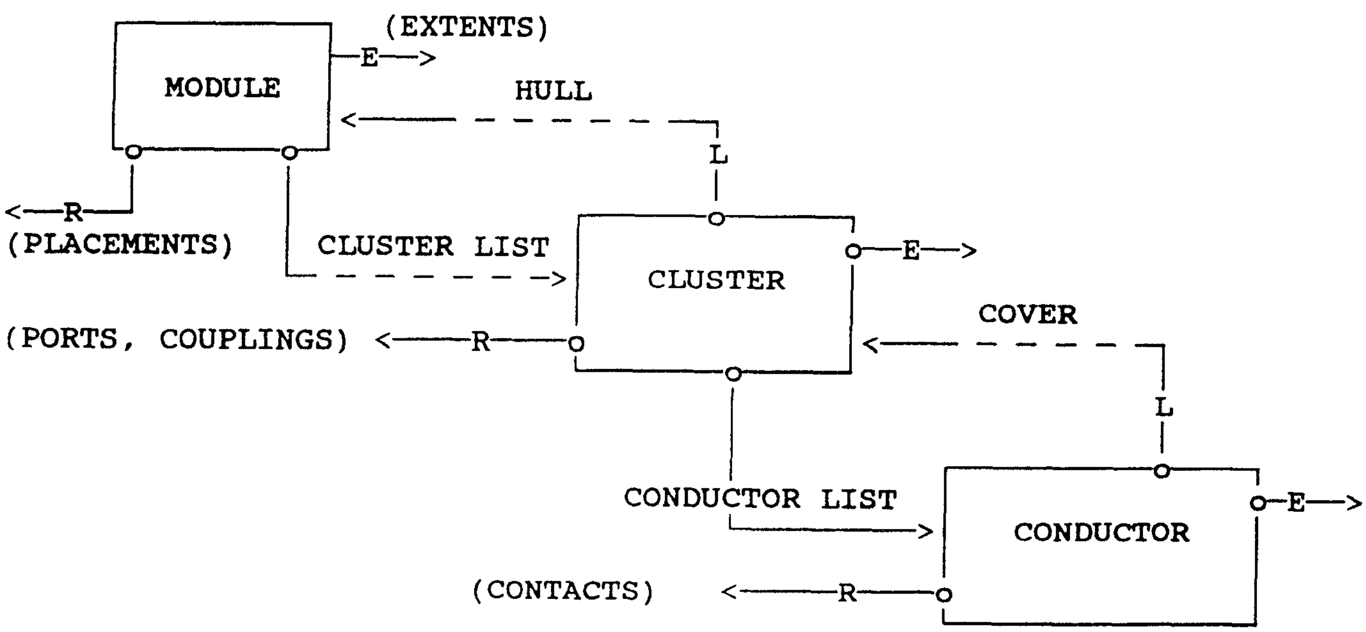


Figure 5. Diagram Representation of Module Object (Ref. 1)

```
--Relations
    type Relation_Types is ( Port, Contained, Connected,
             Conducted, Control, Coupled, I_Coupled, Stimuli );
    type Relation_Record( Relation_Type : Relation_Types );
    type Pointer_To_Relation_Record is access Relation_Record;


    type Relation_Record( Relation_Type : Relation_Types ) is
     record
      Object_A : Pointer_To_Object_Record   := null;
      Next_A   : Pointer_To_Relation_Record := null;
      Object_B : Pointer_To_Object_Record   := null;
      Next_B   : Pointer_To_Relation_Record := null;

      Extents  : Pointer_To_Extent_Record   := null;
      case Relation_Type is
        when contained => Placement     :
                              Pointer_To_Placement_Record;
        when connected => Contact_Type  :
                              Technology.Class_Member_Indices;
        when control   => Transistor    :
                              Pointer_To_Transistor_Record;

        when conducted => Diode_Type    :
                              Technology.Class_Member_Indices;
        when coupled   => Capacitance   : Real := 0.0;
        when stimuli   => Stimuli       :
                          Data_Types.Pointer_To_Stimuli_Record;
        when port      => Terminal      :
                              Pointer_To_Terminal_Record;
        when I_coupled => Instance      :
                              Pointer_To_Coupling_Record;
      end case;
   end record;
```

Figure 6. Example of CADET RELATIONS OBJECT

```
Procedure PARSE_Spice_File ( Name :
                              in dynamic_strings.dyn_string) is

      ------------- declare variables
 begin
 Open (Spice_Ascii_file,Name );
 loop--on lines, exit if end of file
     -- get data string from ascii file    the next line
     -- get first word from string
     -- Check for beginning of signal data -  #TIME
     if Match( First_Word, "Sample" ) then
        Get_Word(Data);-- get second word and convert to integer
        Integer_text_io.get(Data.s
               (Data.wfc..(Data.wlc)),Sample_number,Last);
     elsif Match( First_word, "Time") then
         -- get next line in file
       loop--columns, total of 5, first time , the rest voltage
        get_word (Data);--first signal names, then load record
        t_Wave(Next):= new data_types.waveform_record
                 (Sample_Number);
        t_Wave(Next).Class := Predicted ;
        t_Wave(Next).Name :=
         Dynamic_strings.D_string (Data.S(Data.wfc..Data.wlc));
        Name_Exists := Next_Word_Exists (Data);
        if not (name_exists) then
           exit;
        end if; -- no more signal names
        Next := Next + 1;
       end loop;
       for s in 1..Sample_Number
         loop -- now get time and voltage and load in record
          Data:= Get_Data (Spice_ascii_file);--get Next line
           Get_Word (data); -- get word pick up time
            Time := Convert_to_time (data);
            for w in 1..Next
             loop
               Get_Word (data); -- get word pick up voltage
               Voltage := Convert_to_voltage (data);
               t_wave( w ).Samples( s ).Voltage := Voltage;
               t_wave( w ).Samples( s ).Time := Time;
             end loop;
          end loop;
          Unload( t_wave,Waveform_list );
         end if;
      end loop;
    text_io.close (Spice_ascii_file);
    wave_window.accept_a_wave (waveform_list);
    Dispose(Waveform_List);
   end PARSE_Spice_file;
```

Figure 7. CADET SPICE File Parser