GRASP/Ada

Graphical Representations of Algorithms, Structures, and Processes for Ada

Update of GRASP/Ada Reverse Engineering Tools For Ada

Final Report

Delivery Order No. 13
Basic NASA Contract No. NAS8-39131

Department of Computer Science and Engineering
Auburn University, AL 36849-5347

Contact: James H. Cross II, Ph.D.
Principal Investigator
(205) 844-4330
cross@eng.auburn.edu

December 31, 1992

NASA
NATIONAL AERONAUTICS & SPACE ADMINISTRATION

Report Documentation Page

1. REPORT NO.	2. GOVERNMENT ACCESSION NO	3. RECIPIENT'S CATALOG NO
4. TITLE AND SUBTITLE Update of GRASP/ADA Reverse Engineering Tools for Ada		5. REASON DATE December 31, 1992
		6. PERFORMING ORGANIZATION CODE CSE, Auburn University
7. AUTHOR(S) James H. Cross II, Ph.D.		8. PERFORMING ORGANIZATION REPORT NO. CSE 92-10
		10. WORK UNIT NO. Delivery Order No. 13
9. PERFORMING ORGANIZATION NAME AND ADDRESS Auburn University Computer Science and Engineering Auburn University, AL 36849-5347		11. CONTRACT OR GRANT NO. Basic NASA Contract No. NAS 8-39131
		13. TYPE OF REPORT AND PERIOD COVERED Final Report July 1-December 31, 1992
12. SPONSORING AGENCY NAME AND ADDRESS National Aeronautics and Space Administration Washington, D.C. 20546-0001 Marshall Space Flight Center, MSFC, AL 35812		14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

16. ABSTRACT

Update of prototype created by the GRASP/Ada project (Graphical Representation of Algorithms, Structure and Processes), which generates control structure diagrams from Ada source code.

17. KEY WORDS (SUGGESTED BY AUTHORS) Reverse Engineering, Ada Detailed Design, Maintenance	18. DISTRIBUTION STATEMENT Unlimited

19. SECURITY CLASSIF. (OF THIS REPORT)	20. SECURITY CLASSIF. (OF THIS PAGE)	21. NO. OF PAGES	22. PRICE
Unclassified	Unclassified	52	

NASA FORM 1626 OCT 86

Update of GRASP/Ada

Graphical Representations of Algorithms, Structures, and Processes for Ada

Reverse Engineering Tools For Ada

Final Report

Delivery Order No. 13
Basic NASA Contract No. NAS8-39131

James H. Cross II, Ph.D.
Principal Investigator

December 31, 1992

Abstract

The GRASP/Ada project (*Graphical Representations of Algorithms, Structures, and Processes for Ada*) has successfully created and prototyped a new algorithmic level graphical representation for Ada software, the Control Structure Diagram (CSD). The primary impetus for creation of the CSD was to improve the comprehension efficiency of Ada software and, as a result, improve reliability and reduce costs. The emphasis has been on the automatic generation of the CSD from Ada PDL or source code to support reverse engineering and maintenance. The CSD has the potential to replace traditional prettyprinted Ada source code. In Phase 1 of the GRASP/Ada project, the CSD graphical constructs were created and applied manually to several small Ada programs. A prototype (Version 1) was designed and implemented using FLEX and BISON running under VMS on a VAX 11-780. In Phase 2, the prototype was improved and ported to the Sun 4 platform under UNIX. A user interface was designed and partially implemented using the HP widget toolkit and the X Windows System. In Phase 3, the user interface was extensively reworked using the Athena widget toolkit and X Windows. The prototype was applied successfully to numerous Ada programs ranging in size from several hundred to several thousand lines of source code. Following Phase 3, the prototype was evaluated by software engineering students at Auburn University and then updated with significant enhancements to the user interface including editing capabilities. Version 3.2 of the prototype has been prepared for limited distribution to facilitate further evaluation. The current prototype provides the capability for the user to generate CSDs from Ada PDL or source code in a reverse engineering as well as forward engineering mode with a level of flexibility suitable for practical application.

ACKNOWLEDGEMENTS

We appreciate the assistance provided by NASA personnel, especially Mr. Keith Shackelford whose guidance has been of great value. The following is an alphabetical listing of the team members who have participated in various phases of the project. An asterisk (*) indicates the team member worked on the Update of GRASP/Ada addressed in this report.

Principal Investigator:

Dr. James H. Cross II, Principal Investigator*

Graduate Research Assistants:

Richard A. Davis
Charles H. May
Kelly I. Morrison*
Timothy A. Plunkett
Narayana S. Rekapalli*
Darren Tola

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 Introduction

Computer professionals have long promoted the idea that graphical representations of software can be extremely useful as comprehension aids when used to supplement textual descriptions and specifications of software, especially for large complex systems [SHU88, AOY89, SCA89]. The general goal of this research has been the investigation, formulation and generation of *graphical representations of algorithms, structures, and processes for Ada* (GRASP/Ada). This specific task has focused on *reverse engineering* of control structure diagrams from Ada PDL or source code.

Reverse engineering normally includes the processing of source code to extract higher levels of abstraction for both data and processes. The primary motivation for reverse engineering is increased support for software reusability, verification, and software maintenance, all of which should be greatly facilitated by automatically generating a set of "formalized diagrams" to supplement the source code and other forms of existing documentation. The overall goal of the GRASP/Ada project is to provide the foundation for a CASE (computer-aided software engineering) environment in which reverse engineering and forward engineering (development) are tightly coupled. In this environment, the user may specify the software in a graphically-oriented language and then automatically generate the corresponding Ada code [ADA83]. Alternatively, the user may specify the software in Ada or Ada/PDL and then automatically generate the graphical representations either dynamically as the code is entered or as a form of post-processing.

The GRASP/Ada project was divided into three primary development phases followed by an evaluation and update phase. Each of these phases is briefly described below.

1.1 Phase 1 - The Control Structure Diagram For Ada

Phase 1 focused on a survey of graphical notations for software with concentration on detailed level diagrams such as those found in [MAR85, TRI89], and the development of a new algorithmic or PDL/code level diagram for Ada. Tentative graphical control constructs for the *Control Structure Diagram* (CSD) were created and initially prototyped in a VAX/VMS environment. This included the development of special diagramming fonts for both the screen and printer and the development of parser and scanner using UNIX based tools such as LEX and YACC. The CSD is described in Section 2.0.

1.2 Phase 2 - The GRASP/Ada Prototype and User Interface

During Phase 2, the prototype was extended and ported to a Sun/UNIX environment. The development of a user interface based on the X Window System represented a major part of the extension effort. Verdix Ada and the Verdix DIANA interface were acquired as potential commercial tools upon which to base the GRASP/Ada prototype. Architectural diagrams for Ada were surveyed and the OOSD notation [WAS89] was identified as having good potential for accurately representing many of the varied architectural features of an Ada software system. Phase 2 also included the preliminary design and a separate exploratory prototype for an architectural CSD. The best aspects of architectural CSD are expected to be integrated into the fully operational GRASP/Ada prototype during a future phase of the project.

1.3 Phase 3 - CSD Generation Prototype and Preliminary Object Diagram Prototype

Phase 3 has had two major thrusts: (1) completion of an operational GRASP/Ada prototype which generates CSDs and (2) the development of a preliminary prototype which generates object diagrams directly from Ada source code. Completion of the GRASP/Ada CSD prototype (CSDgen) included the addition of substantial functionality, via the User Interface, to make the prototype easier to use. CSDgen was installed and demonstrated on a Sun workstation at Marshall Space Flight Center, Alabama.

The development of a preliminary prototype for generating architectural object diagrams (ODgen) for Ada source/PDL was an effort to determine feasibility rather than to deliver an operational prototype as was the case with CSD generator above. The preliminary prototype has indicated that the development of the components to recover the information to be included in the diagram, although a major effort, is relatively straightforward. However, the research has also indicated that the major obstacle for automatic object diagram generation is the automatic layout of the diagrams in a human readable and/or aesthetically pleasing format. A user extensible rule base, which automates the diagram layout task, is expected to be formulated during future GRASP research. Interactive Development Environment's Software through Pictures (IDE/StP), which supports the OOSD notation in a forward engineering sense, has been identified as a candidate for a commercial CASE environment with which to integrate GRASP/Ada reverse engineering system.

1.4 Update of the GRASP/Ada

Following Phase 3, the Version 3.1 prototype was used in several software engineering classes at Auburn University, evaluated, and enhanced to create Version 3.2. The following tasks were performed during the current effort as an update to GRASP/Ada.

(1) **The Graphical Representation of Algorithm, Structures, and Processes (GRASP) Ada tool was evaluated and modified.** As part of the ongoing evaluation of GRASP/Ada, GRASP/Ada was used in CSE 422 (Introduction to Software Engineering). An evaluation instrument was developed and administered to collect feedback from the students prior to widespread release to academic, business, and industrial communities. As a result of the evaluation, numerous modifications and enhancements were made to the User Interface. The evaluation is described in Section 6.0.

(2) **The *work in progress* of the GRASP Ada evaluation and modification were presented at the Reverse Engineering Forum, Burlington, MA, September 15-17, 1992.**

(4) **The UNIX Command Set was updated to reflect changes to the prototype tool.** In particular, makefiles were streamlined to make recompilation and installation more straightforward.

(5) **The Man Page and Getting Started were written for the GRASP prototype.** Although the window-based User Interface is relatively intuitive, one of items requested most by the students that evaluated the prototype was a User Manual. While *Getting Started* (see Appendix A) and the *Man Page* provide necessary user information, a formal user manual is expected to be developed as part of a future GRASP/Ada update.

(3) **GRASP/Ada was extended to facilitate use with the CASE tool, "Software Through Pictures" from Interactive Development Environments, Inc.** The prototype was modified so that it could be invoked from StP with a pspec or PDL file. Appendix B provides a description of this procedure.

(6) **The GRASP Ada prototype was prepared for limited distribution via the network.** To date, over 200 requests for information regarding GRASP/Ada have been received as a result of publications generated from this research. Responding to these requests is an important element of the ongoing evaluation and refinement of the GRASP/Ada reverse engineering tool.

The following sections describe the control structure diagram, the GRASP/Ada system model, the user interface, the control structure diagram generator, evaluation of the CSD and prototype, and future requirements. The overall rationale for the development of the CSD is described in [CRO90a, CRO90b], which were written during Phase 1. A taxonomy and extensive literature review of reverse engineering can be found in [CHI90, CRO92], which were written during Phases 2 and 3.

2.0 The Control Structure Diagram

Advances in hardware and software, particularly high-density bit-mapped monitors and window-based user interfaces, have led to a renewed interest in graphical representation of software. Although much of the research activity in the area of software visualization and computer-aided software engineering (CASE) tools has focused on architectural-level charts and diagrams, the complex nature of the control constructs and control flow defined by programming languages such as Ada and C and their associated PDLs, makes source code and detailed design specifications attractive candidates for graphical representation. In particular, source code should benefit from the use of an appropriate graphical notation since it must be read many times during the course of initial development, testing and maintenance. The control structure diagram (CSD) is a notation intended specifically for the graphical representation of algorithms in detailed designs as well as actual source code. The primary purpose of the CSD is to reduce the time required to comprehend software by clearly depicting the control constructs and control flow at all relevant levels of abstraction. The CSD is a natural extension to existing architectural graphical representations such as data flow diagrams, structure charts, and object diagrams.

The CSD, which was initially created for Pascal/PDL [CRO88], has been extended significantly so that the graphical constructs of the CSD map directly to the constructs of Ada. The rich set of control constructs in Ada (e.g. task rendezvous) and the wide acceptance of Ada/PDL by the software engineering community as a detailed design language made Ada a natural choice for the basis of a graphical notation. A major objective in the philosophy that guided the development of the CSD was that the graphical constructs should supplement the code and/or PDL without disrupting their familiar appearance. That is, the CSD should appear to be a natural extension to the Ada constructs and, similarly, the Ada source code should appear to be a natural extension of the diagram. This has resulted in a concise, compact graphical notation which attempts to combine the best features of diagraming with those of well-indented PDL or source code.

2.1 Background

Graphical representations have been recognized as having an important impact in communicating from the perspective of both the "writer" and the "reader." For software, this includes communicating requirements between users and designers and communicating design specifications between designers and implementors. However, there are additional areas where the potential of graphical notations have not been fully exploited. These include communicating the semantics of the actual implementation represented by the source code to personnel for the purposes of testing and maintenance, each of which are major resource sinks in the software life cycle. In particular, Selby [SEL85] found that code reading was the most cost effective method of detecting errors during the verification process when compared to functional testing and structural testing. And Standish [STA85] reported that program understanding may represent as much as 90% of the cost of maintenance. Hence, improved comprehension efficiency resulting from the integration of graphical notations and source code could have a significant impact on the overall cost of software production.

Since the flowchart was introduced in the mid-50's, numerous notations for representing algorithms have been proposed and utilized. Several authors have published notable books and papers that address the details of many of these [MAR85, TRI88, SHN77]. Tripp, for example, describes 18 distinct notations that have been introduced since 1977 and Aoyama et.al. describes the popular diagrams used in Japan. In general, these diagrams have been strongly influenced by structured programming and thus contain control constructs for sequence, selection, and iteration. In addition, several contain explicit EXIT structures to allow single entry / multiple exit control flow through a block of code, as well as PARALLEL or concurrency constructs. However, none the diagrams cited explicitly contains all of the control constructs found in Ada.

Graphical notations for representing software at the algorithmic level have been neglected, for the most part, by business and industry in the U.S. in favor of non-graphical PDL. A lack of automated support and the results of several studies conducted in the seventies which found no significant difference in the comprehension of algorithms represented by flowcharts and pseudo-code [SHN77] have been a major factors in this underutilization. However, automation is now available in the form of numerous CASE tools and recent empirical studies reported by Aoyami [AOY89] and Scanlan [SCA89] have concluded that graphical notations may indeed improve the comprehensibility and overall productivity of software. Scanlan's study involved a well-controlled experiment in which deeply nested if-then-else constructs, represented in structured flowcharts and pseudo-code, were read by intermediate-level students. Scores for the flowchart were significantly higher than those of the PDL. The statistical studies reported by Aoyami et.al. involved several tree-structured diagrams (e.g., PAD, YACC II, and SPD) widely used in Japan which, in combination with their environments, have led to significant gains in productivity. The results of these recent studies suggest that the use of a graphical notation with appropriate automated support for Ada/PDL and Ada should provide significant increases productivity over current non-graphical approaches.

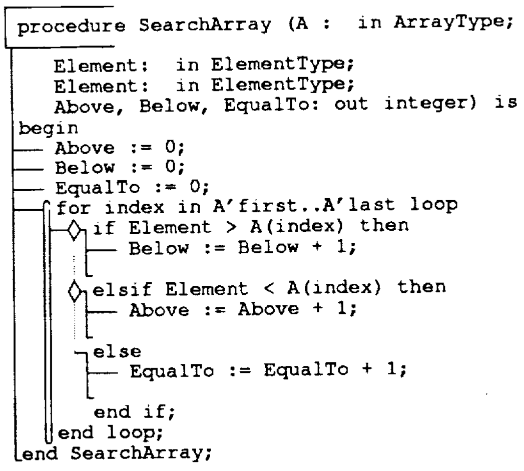
2.2 The Control Structure Diagram Illustrated

Two examples are presented below to illustrate the CSD. The first shows the basic control constructs of sequence, selection and iteration in Ada. These three control constructs are common to all structured procedural languages such as Ada, C, and Pascal. The second example illustrates a more complex control construct, the task rendezvous in Ada.

Figure 1 contains an Ada procedure called SearchArray that searches an array A of elements and counts the number of elements above, below, and/or equal to a specified element. Figure 2 contains the CSD for SearchArray which includes the three basic control constructs sequence, selection, and iteration. Although this is a very simple example, the CSD clearly indicates the levels of control inherent in the nesting of control statements. For example, at level 1 there are four statements executed in sequence - the three assignment statements and the *for* loop. The *for* loop defines a second level of control which contains a single statement, the *if* statement, which in turn defines three separate level 3 sequences, each of which contains one assignment statement. It is noteworthy that even the CSDs for most production strength procedures rarely contain more than ten statements at level 1 or in any of the subsequences defined by control constructs for selection and iteration. This graphical chunking on the basis of functionality and level of control appears to have a substantial positive effect on detailed comprehension of the software.

```
procedure SearchArray (A :  in ArrayType;
    Element:  in ElementType;
    Above,Below, EqualTo: out integer)   is

begin
   Above := 0;
   Below := 0;
   EqualTo := 0;
   for index in A'first..A'last loop
      if Element > A(index) then
         Below := Below + 1;

      elsif Element < A(index) then
         Above := Above + 1;

      else
         EqualTo := EqualTo + 1;

      end if;
   end loop;
end SearchArray;
```

Figure 1. Ada Source Code for Procedure SearchArray.



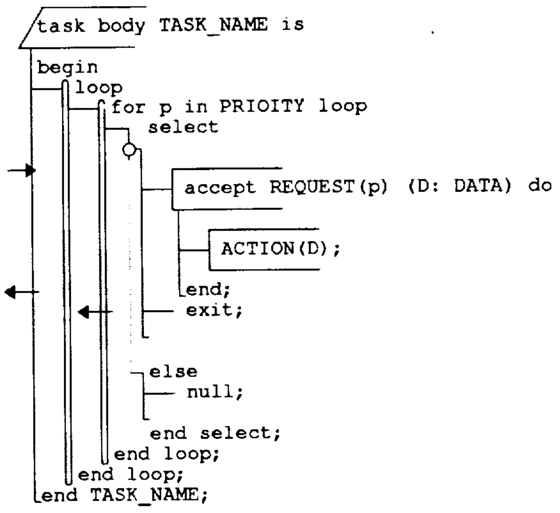
```
procedure SearchArray (A :  in ArrayType;

   Element:  in ElementType;
   Element:  in ElementType;
   Above, Below, EqualTo: out integer) is
begin
   Above := 0;
   Below := 0;
   EqualTo := 0;
   for index in A'first..A'last loop
      if Element > A(index) then
         Below := Below + 1;

      elsif Element < A(index) then
         Above := Above + 1;

      else
         EqualTo := EqualTo + 1;

      end if;
   end loop;
end SearchArray;
```

Figure 2. CSD for Procedure SearchArray.

Figures 3 and 4 contain an Ada task body CONTROLLER adapted from [BAR84], which loops through a priority list attempting to accept selectively a REQUEST with priority P. Upon on acceptance, some action is taken, followed by an exit from the priority list loop to restart the loop with the first priority. In typical Ada task fashion, the priority list loop is contained in an outer infinite loop. This short example contains two threads of control: the rendezvous, which enters and exists at the accept statement, and the thread within the task body. In addition, the priority list loop contains two exits: the normal exit at the beginning of the loop when the priority list has been exhausted, and an explicit exit invoked within the

```
task body TASK_NAME is

begin
   loop
      for p in PRIOITY loop
         select


            accept REQUEST(p)  (D: DATA)  do

               ACTION(D);

            end;
            exit;


         else
            null;

         end select;
      end loop;
   end loop;
end TASK_NAME;
```

Figure 3. Ada Source Code for Task Body Controller.



```
task body TASK_NAME is
begin
   loop
      for p in PRIOITY loop
         select

            accept REQUEST(p)  (D: DATA)  do

               ACTION(D);

            end;
            exit;


         else
            null;

         end select;
      end loop;
   end loop;
end TASK_NAME;
```

Figure 4. CSD for Ada Task Body Controller.

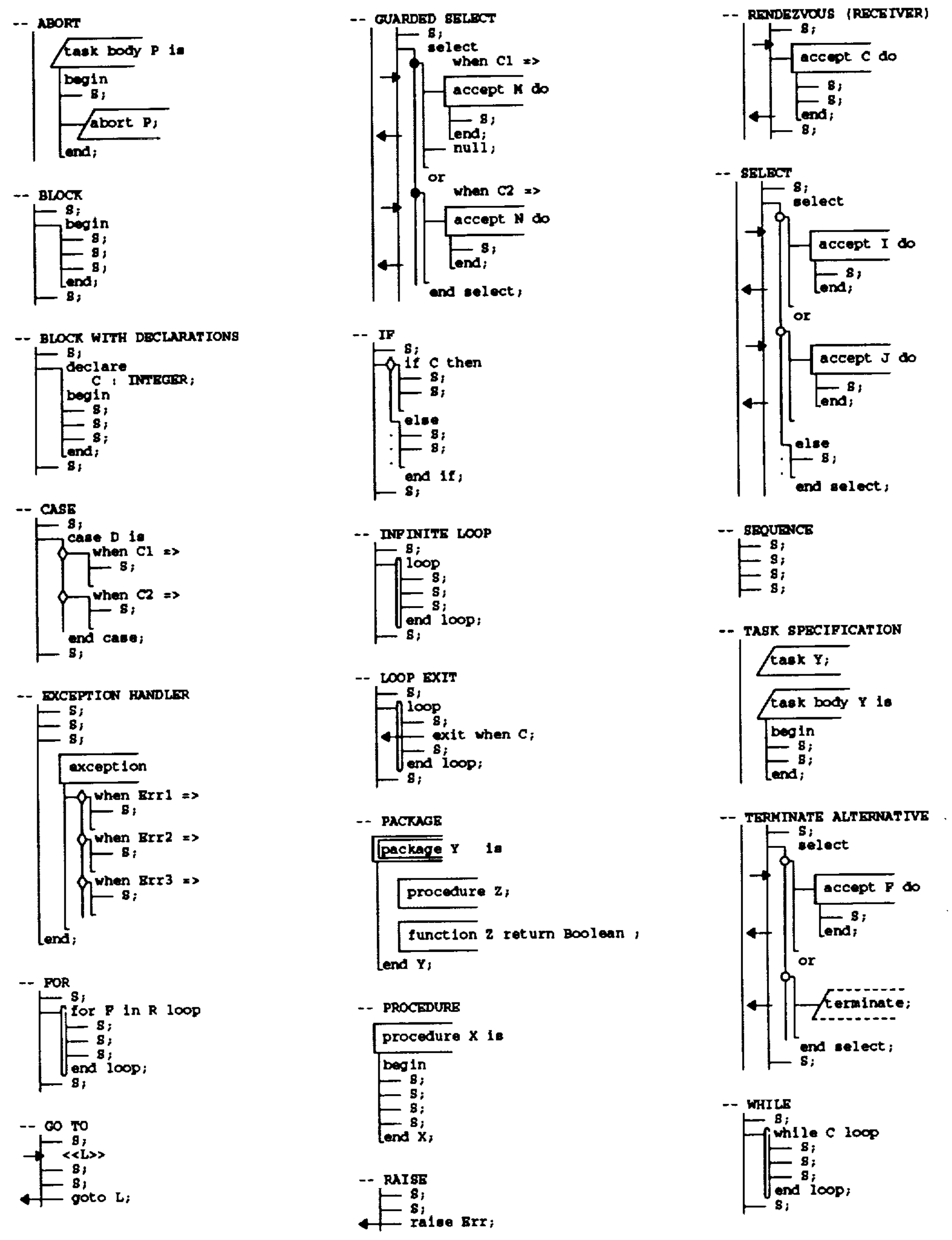
select statement. While the concurrency and multiple exits are useful in modeling the solution, they do increase the effort required of the reader to comprehend the code.

The CSD in Figure 4 uses intuitive graphical constructs to depict the point of rendezvous, the two nested loops, the select statement guarding the accept statement for the task, the unconditional exit from the inner loop, and the overall control flow of the task. When reading the code without the diagram, as shown in Figure 3, the control constructs and control paths are much less visible although the same structural and control information is available. With additional levels of nesting and increased physical separation of sequential components, the visibility of control constructs and control paths becomes increasingly obscure, and the effort required of the reader dramatically increases in the absence of the CSD. Now that the CSD has been briefly introduced, the various CSD constructs for Ada are presented in Figure 5. Each of the CSD constructs should be relatively self-explanatory since the CSD is designed to supplement the semantics of the underlying Ada.

2.3 Observations

The control structure diagram is a new graphical tool which maps directly to Ada and Ada PDL. The CSD offers advantages over previously available diagrams in that it is combines the best features of PDL and code with simple intuitive graphical constructs. The potential of the CSD can be best realized during detailed design, implementation, verification and maintenance. The CSD can be used as a natural extension to popular architectural level representations such as data flow diagrams, object diagrams, and structure charts.

The GASP/Ada prototype, described in the following sections, provides for the automatic generation of the CSD from Ada or Ada PDL



```
-- ABORT
task body P is
begin
   S;
   abort P;
end;

-- BLOCK
S;
begin
   S;
   S;
   S;
end;
S;

-- BLOCK WITH DECLARATIONS
S;
declare
   C : INTEGER;
begin
   S;
   S;
   S;
end;
S;

-- CASE
S;
case D is
   when C1 =>
      S;
   when C2 =>
      S;
end case;
S;

-- EXCEPTION HANDLER
S;
S;
S;
exception
   when Err1 =>
      S;
   when Err2 =>
      S;
   when Err3 =>
      S;
end;

-- FOR
S;
for F in R loop
   S;
   S;
   S;
end loop;
S;

-- GO TO
S;
<<L>>
S;
S;
goto L;

-- GUARDED SELECT
S;
select
   when C1 =>
      accept M do
         S;
      end;
      null;
or
   when C2 =>
      accept N do
         S;
      end;
end select;

-- IF
S;
if C then
   S;
   S;
else
   S;
   S;
end if;
S;

-- INFINITE LOOP
S;
loop
   S;
   S;
   S;
end loop;
S;

-- LOOP EXIT
S;
loop
   S;
   exit when C;
   S;
end loop;
S;

-- PACKAGE
package Y   is
   procedure Z;
   function Z return Boolean ;
end Y;

-- PROCEDURE
procedure X is
begin
   S;
   S;
   S;
   S;
end X;

-- RAISE
S;
S;
raise Err;

-- RENDEZVOUS (RECEIVER)
S;
accept C do
   S;
   S;
end;
S;

-- SELECT
S;
select
   accept I do
      S;
   end;
or
   accept J do
      S;
   end;
else
   S;
end select;

-- SEQUENCE
S;
S;
S;
S;

-- TASK SPECIFICATION
task Y;
task body Y is
begin
   S;
   S;
end;

-- TERMINATE ALTERNATIVE
S;
select
   accept F do
      S;
   end;
or
   terminate;
end select;
S;

-- WHILE
S;
while C loop
   S;
   S;
   S;
end loop;
S;
```

Figure 5. Control Structure Diagram Constructs for Ada.

3.0 The GRASP/Ada System Model

The major system components of the GRASP/Ada system are shown in the block diagram in Figure 6. The **User Interface** was built using the X Window System and includes a special CSD window (modified text editor) and provides general control and coordination among the other components.

The control structure diagram generator, **CSDgen**, inputs Ada PDL or source code and produces a CSD. CSDgen has its own parser/scanner built using FLEX and BISON, successors of LEX and YACC. It also includes its own printer utilities. As such, CSDgen is a self-sufficient component which can be executed from the user interface or the command line without the commercial components. When changes are made to the Ada PDL or source code, the entire file must be reparsed to produce an updated CSD. A CSD editor, which will provide for dynamic incremental modification of the CSD, is currently in the planning stages.

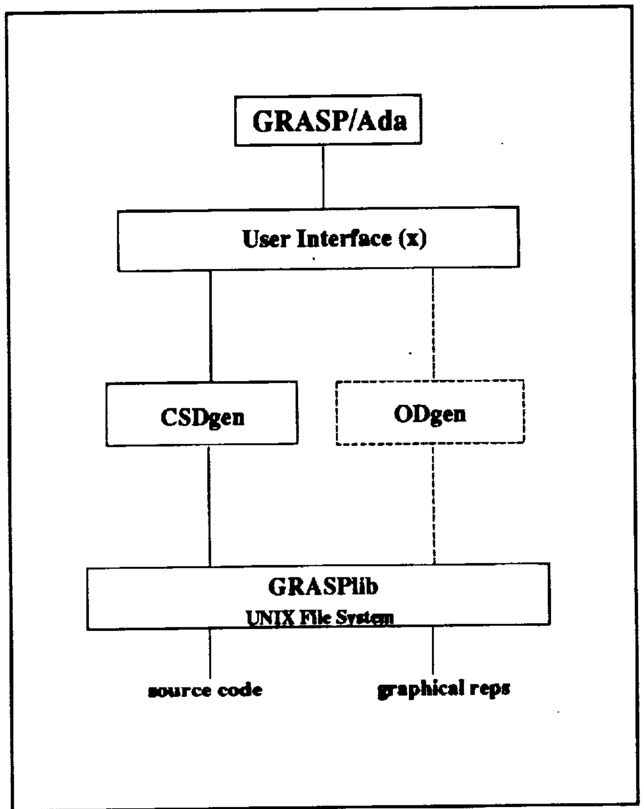


Figure 6. GRASP/Ada System Block Diagram.

The object diagram generation component, **ODgen**, is in the analysis phase and has been implemented as a separate preliminary prototype. The dashed lines indicate future integration. The feasibility of automatic diagram layout remains under investigation. Beyond automatic diagram layout, several design alternatives have been identified. The major alternatives include the decision of whether to attempt to integrate GRASP/Ada directly with commercial components, namely (1) the Verdix Ada development system (VADS) and DIANA interface for extraction of diagram information and (2) IDE's Software through Pictures, Ada Development Environment (IDE/StP/ADE) for the display of the object diagrams.

The GRASP/Ada library component, **GRASPlib**, allows for coordination of all generated items with their associated source code. The current file organization uses standard UNIX directory conventions as well as default naming conventions. For example, all Ada source files end in *.a*, the corresponding CSD files end in *.a.csd*, and the corresponding print files end in *.a.csd.ps*. In the present prototype, library complexity has been keep to a minimum by relying on the UNIX directory organization. Its purpose is to facilitate navigation among the diagrams and the production of sets of diagrams.

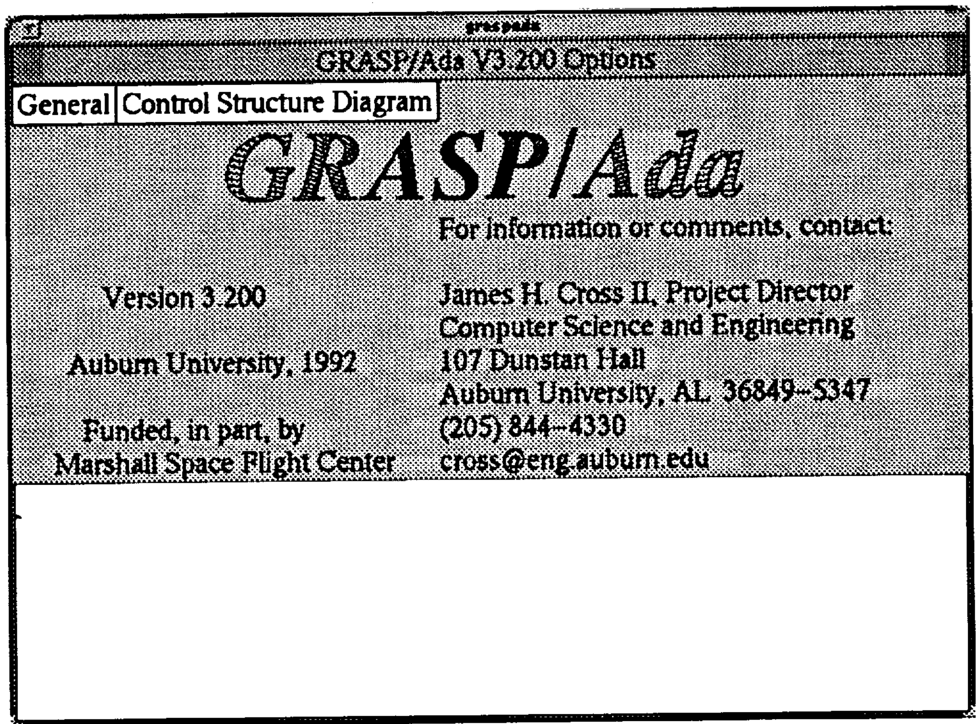
4.0 User Interface

GRASP/Ada user interface was developed using the X Window System, Version 11 Release 4 (X11R4). The X Window System, or simply X, meets the GRASP/Ada user interface requirements of an industry-standard window based environment which supports portable graphical user interfaces for application software. Some of the key features which make X attractive for this application are its availability on a wide variety of platforms, unique device independent architecture, adaptability to various user interface styles, support from a consortium of major hardware and software vendors, and low acquisition cost. With its unique device independent architecture, X allows programs to display windows on any hardware that supports the X Protocol. X does not define any particular user interface style or policy, but provides mechanisms to support many various interface styles.

The specifications and figures that follow are intended to define the look and feel of the GRASP/Ada User Interface as well as indicate much of the current and planned functionality of the CSD generator. The Man Page provides additional information.

4.1 System Window

The System window, shown in Figure 7, provides the user with the overall organization and structure of the GRASP/Ada tool. Option buttons include: General and Control Structure Diagram. These are briefly described below. A future button is planned for Object Diagram.



For information or comments, contact:

Version 3.200

Auburn University, 1992

Funded, in part, by
Marshall Space Flight Center

James H. Cross II, Project Director
Computer Science and Engineering
107 Dunstan Hall
Auburn University, AL 36849–5347
(205) 844–4330
cross@eng.auburn.edu

Figure 7. GRASP/Ada System Window.

General - provides for selection of a printer access to the user manual (see Figure 8).

Control Structure Diagram - allows the user to open one or more CSD windows, close all CSD windows that are currently open, and generate CSDs in a batch mode (see Figure 9). In addition, a list of all CSD windows currently open is presented to the user.

Since GRASP/Ada is expected to be used to process and analyze large existing Ada software systems consisting of perhaps hundreds of files, the option to generate a set of CSDs in batch mode is particularly useful. Generating a set of CSDs is facilitated by entering *.*a* or some other wildcard with a conventional source file extension, for the file name. A CSD generation **summary window** displays the progress of the generation by listing each file as it is being processed and any resulting error messages. The summary concludes with number of files processed and the number of errors encountered. The default for each CSD file name is the source file name with **.csd** appended. If an error is encountered, an extension of **.err** is used. As the CSDs are generated, the GRASP library is updated, which currently consists of populating a specified directory with file images of the CSDs. Generating a set of CSDs can be considered a user interface requirement rather than strictly a CSD generator requirement.

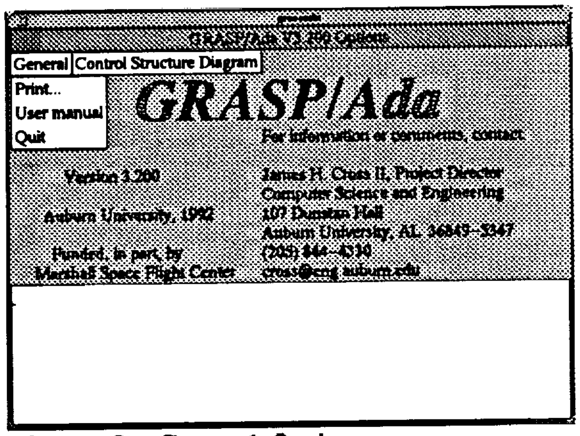

GRASP/Ada V3.200 Options
General
Control Structure Diagram
Print...
User manual
Quit
GRASP/Ada
For information or comments, contact
Version 3.200
Auburn University, 1992
Funded, in part, by
Marshall Space Flight Center
James H. Cross II, Project Director
Computer Science and Engineering
107 Dunstan Hall
Auburn University, AL 36849-5347
(205) 844-4330
cross@eng.auburn.edu

Figure 8. General Options.

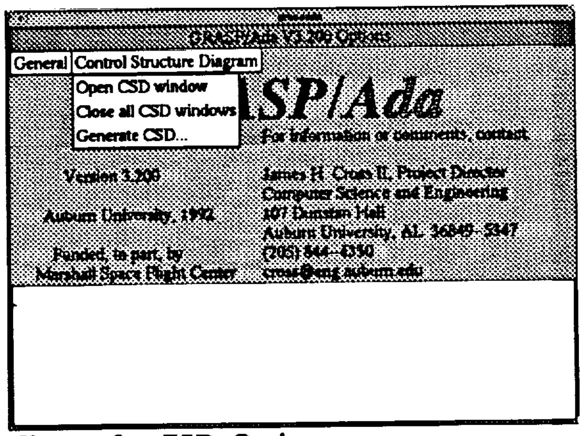


Figure 9. CSD Options.

4.2 Control Structure Diagram Window

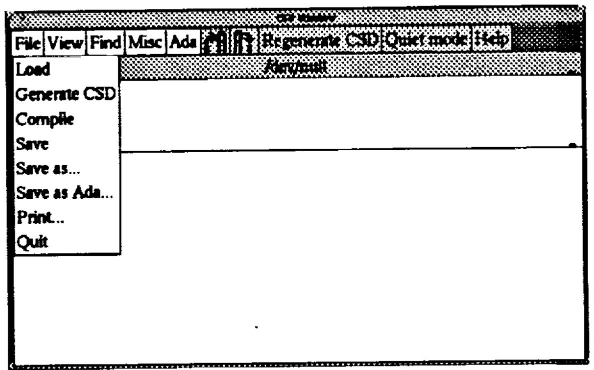
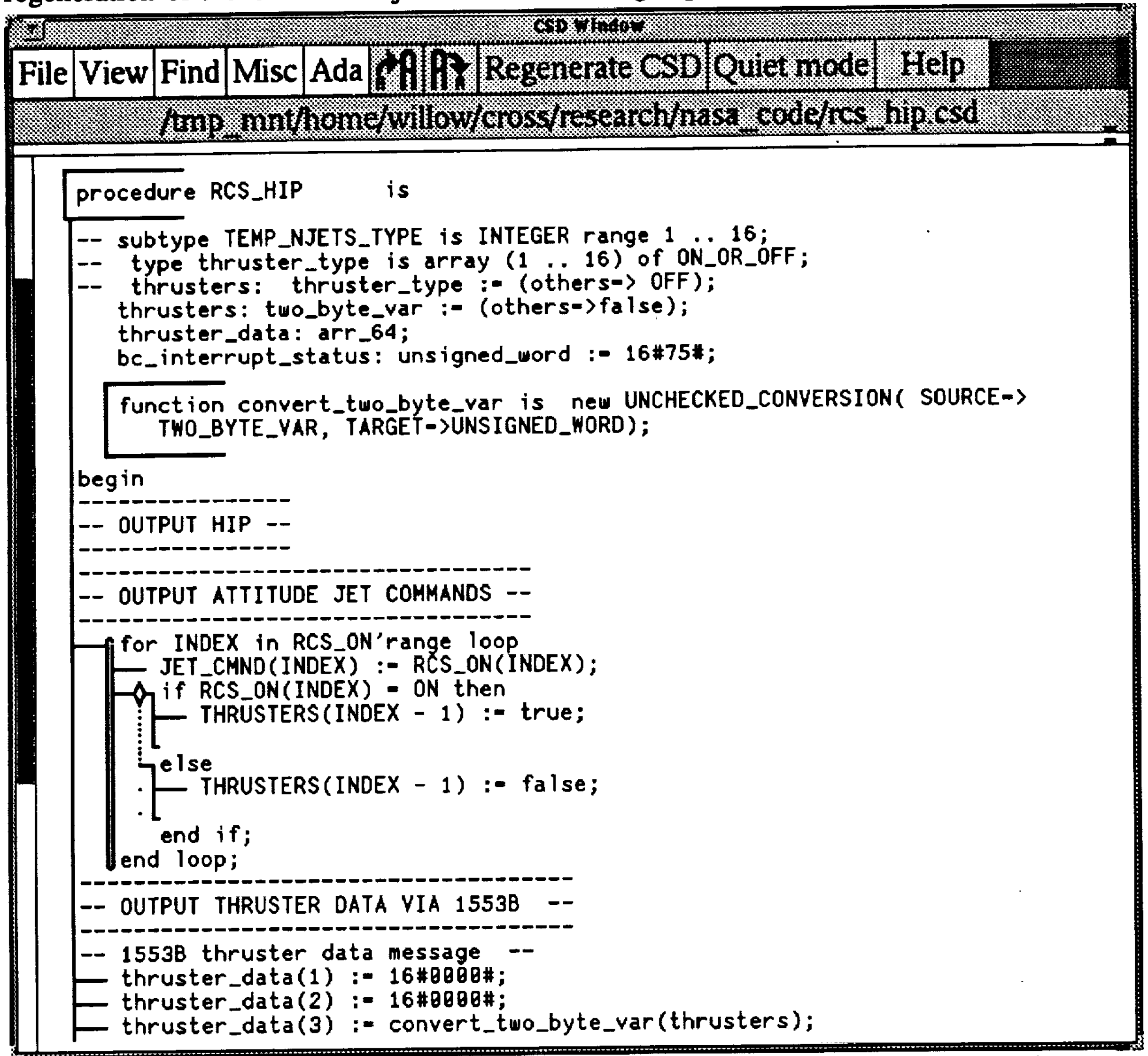


Figure 10. CSD Window File Options.

The CSD window, shown in Figure 10 with file options displayed, provides the user with capabilities for generating and viewing a CSD for an Ada PDL or source file. Multiple CSD windows may be opened to access several CSD files at once. CSD file names and their associated directory paths are selected under the File option and displayed at the top of each window. Figure 11 shows a CSD window after a procedure provided by

NASA has been loaded and the CSD generated by clicking **Regenerate CSD** on the menu selection bar or by clicking **Generate CSD** under **File** options. In the current version of GRASP/Ada, generation of the CSD is done on a file-level basis where each file contains one or more units. When changes are made to the source code, the entire CSD for the file involved is regenerated. Future versions of GRASP/Ada will address incremental regeneration of the CSD in conjunction with editing capabilities in the CSD window.



```
procedure RCS_HIP      is
-- subtype TEMP_NJETS_TYPE is INTEGER range 1 .. 16;
--  type thruster_type is array (1 .. 16) of ON_OR_OFF;
--  thrusters:  thruster_type := (others=> OFF);
   thrusters: two_byte_var := (others=>false);
   thruster_data: arr_64;
   bc_interrupt_status: unsigned_word := 16#75#;

   function convert_two_byte_var is  new UNCHECKED_CONVERSION( SOURCE->
      TWO_BYTE_VAR, TARGET->UNSIGNED_WORD);

begin
----------------
-- OUTPUT HIP --
----------------
-----------------------------------
-- OUTPUT ATTITUDE JET COMMANDS --
-----------------------------------
   for INDEX in RCS_ON'range loop
      JET_CMND(INDEX) := RCS_ON(INDEX);
      if RCS_ON(INDEX) = ON then
         THRUSTERS(INDEX - 1) := true;

      else
         THRUSTERS(INDEX - 1) := false;

      end if;
   end loop;
--------------------------------------
-- OUTPUT THRUSTER DATA VIA 1553B  --
--------------------------------------
-- 1553B thruster data message  --
   thruster_data(1) := 16#0000#;
   thruster_data(2) := 16#0000#;
   thruster_data(3) := convert_two_byte_var(thrusters);
```

Figure 11. CSD Window with Procedure Provided by NASA After CSD Generation.

The CSD window Options

File - allows the user to select from numerous options including:

Load - loads a CSD file. A window is presented that allows the user to navigate among directories and select a file.

Generate CSD - generates a CSD from source code or to regenerate a CSD after modification. When the CSD window is opened and loaded with a source file without a *.csd* extension, a separate CSD window is automatically opened to display the CSD when it is generated. Note that CSD graphics characters, if any, are filtered prior to the parse or reparse. Currently, this option is the same as **Regenerate CSD** on the menu selection bar of the CSD window (described below).

Compile - is a future option to allow an Ada compiler to be called from the CSD window. The alternative is to have GRASP/Ada called as an editor from an Ada development tool such as Sun's AdaVision.

Save - saves the CSD file with the same name as was loaded.

Save as ... - saves the CSD file with a new name.

Save as Ada - filters the CSD characters from the CSD file and writes to a file with a *.a* extension.

Print - presents a window which allows the user to select various print options such as point size, page numbers, and header, and then generates a PostScript file (.ps) from the .csd file and sends it to the selected printer.

Quit - closes the CSD window.

View - currently allows the user to select one of several window fonts (also see option AA AA below). Future options may include the following.

Enable Collapse {Disable Collapse} - will allow the user to collapse the CSD based on its control constructs.

Suppress CSD {Show CSD} - will allow the user to suppress or hide the CSD giving the appearance of prettyprinted code.

Open TOC Window - will access the GRASP library and displays a table of contents based on Ada scoping.

Open Index Window - will access the GRASP library and display an index of units in alphabetical order.

Find - (not activated) allows the user to perform search and replace operations. A Search window can be opened by pressing Ctrl-S.

Misc - allows the user to *show* or *hide* The CSD character panel. With the panel visible, CSD characters can be inserted directly into the current window, primarily for the purpose of experimentation.

Ada - displays Ada control constructs and enables the user to insert them directly into the current window at the location of the curser (see Figure 12). A syntactically correct program can be constructed quickly using this option. Figure 13 shows a program structure resulting from four clicks on the Ada constructs: *procedure body*, *while loop*, *if/then/else*, and *for loop*. The template placeholders can be modified or replaced as necessary.

AA AA - allows the user to increase or decrease the font size for the current window, thus shrinking or expanding the overall size of the CSD.

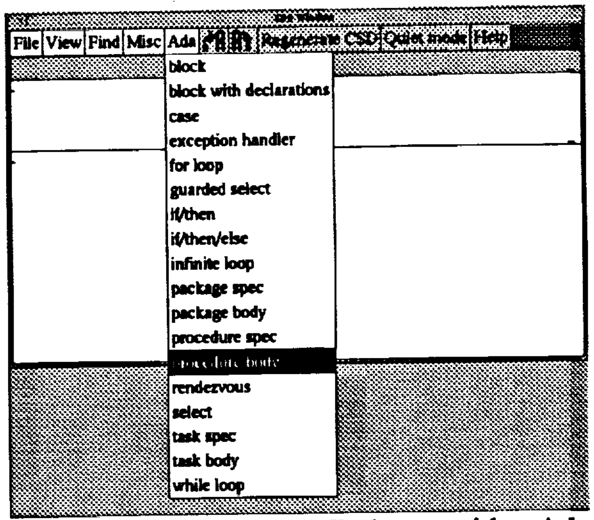
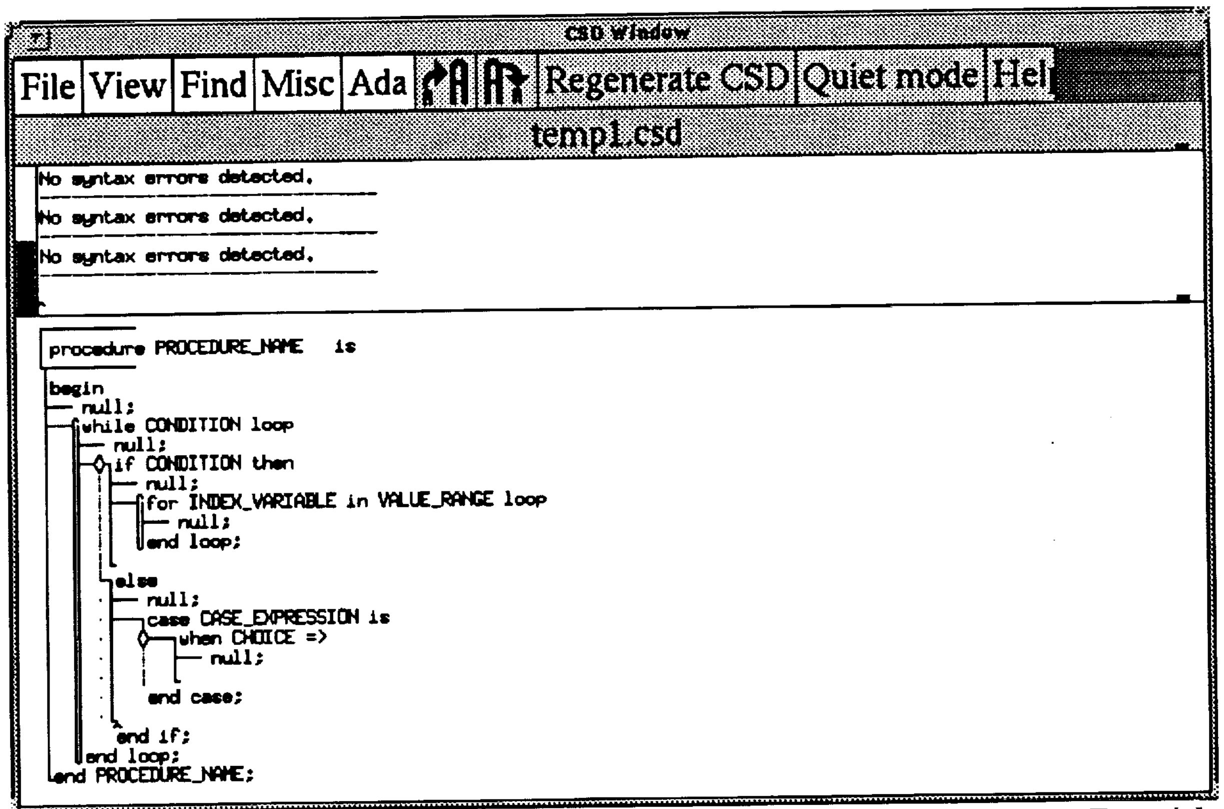


Figure 12. CSD Window with Ada Constructs - *procedure body* selected.



```
No syntax errors detected.
No syntax errors detected.
No syntax errors detected.
```

```
procedure PROCEDURE_NAME   is
begin
  null;
  while CONDITION loop
    null;
    if CONDITION then
      null;
      for INDEX_VARIABLE in VALUE_RANGE loop
        null;
      end loop;
    else
      null;
      case CASE_EXPRESSION is
        when CHOICE =>
          null;
      end case;
    end if;
  end loop;
end PROCEDURE_NAME;
```

Figure 13. CSD Window with Program Structure Resulting from Clicking on Four Ada Constructs: *procedure body*, *while loop*, *if/then/else*, and *for loop*.

Regenerate CSD - allows the user to quickly regenerate a CSD after making changes. Currently, this option is the same as **Generate CSD** on the **File** submenu. See above for details.

Quiet mode / Verify mode - allows the user to toggle between to modes of regeneration. Quiet mode assumes that the existing files should be overwritten during generation or regeneration, and Verify mode queries the user before continuing.

4.3 User Interface Summary

The User Interface is expected to continue to evolve, especially as new functionality is added. In particular, implementation based on alternate widget sets is under consideration as well as utilization with other window manager. The requirements definition and design of the current version were done in a learning mode under a schedule that required an operational prototype to be implemented quickly. As a result, many of the features, such as placement of options, are expected to be streamlined considerably. However, the current prototype is suitable for limited practical application, and information collected from current users is expected to have a positive effect on the overall evolution of the prototype.

5.0 Control Structure Diagram Generator

The GRASP/Ada control structure diagram generator (CSDgen) is described in this section from a technical and developmental perspective. A more complete history and rationale for the development of the CSD is contained in [CRO90a, CRO90b]. The graphical constructs produced by CSDgen are summarized in Figure 5 (Section 2.0). Examples of the CSD are presented in conjunction with the User Interface in Section 4.0.

5.1 Generating the CSD

The primary function of CSDgen is to produce a CSD for a corresponding Ada source or PDL file. Although a complete parse is done during CSD generation, CSDgen assumes the Ada source code has been previously compiled and thus is syntactically correct. Currently, little error recovery is attempted when a syntax error is encountered. The diagram is simply generated down to the point of the error. In the case of Ada PDL, non-Ada statements must be valid Ada identifiers so that they are treated like procedure calls. For example, the PDL for "search array for largest element" could be represented as Search_array_for _largest_element.

The current CSDgen prototype builds the diagram directly during the parse by inserting CSD graphics characters into a file along with text. To increase efficiency and improve extensibility, future versions of the CSDgen prototype may use a more abstract intermediate representation.

5.2 Displaying the CSD - Screen and Printer

Basic display capabilities to the screen and printer were implemented during Phase 2. Screen display is facilitated by sending the CSD file to a CSD window opened under an X Window manager. Printing is accomplished by converting the CSD file to a PostScript file and then sending it to a printer. Moving to a more abstract intermediate representation in future versions would necessitate the development of a new set of display routines which will be X Window System based.

CSD Screen Fonts. The default CSD screen font is a bitmap 13 point Courier to which the CSD graphic characters have been added. The font was defined as a bitmap distribution font (BDF) then converted to SNF format required by the X Window System. Four additional screen fonts ranging from 5 to 18 point are user selectable.

CSD Printer Fonts. CSD Printer fonts were initially developed for the HP LaserJet series. These were then implemented as PostScript type 3 fonts and all subsequent font development has been directed towards the PostScript font. The PostScript font provides the most flexibility since its size is user selectable from 1 to 300 points.

5.3 Displaying the CSD - Future Considerations

Layout/Spacing. The general layout of the CSD is highly structured by design. However, the user should have control over such attributes as horizontal and vertical spacing and the optional use of some diagramming symbols. In the current Version 3 CSDgen prototype, horizontal and vertical spacing are not user selectable. They are a part of the CSD file generation and are defaulted to single spacing with 80 characters per line. In order to change these, e.g., from single to double spacing, the CSD file would have to be regenerated. In future versions of the prototype, these options are expected to be handled by the new display routines and, as such, can be modified dynamically without regenerating the CSD file.

Vertical spacing options will include single, double, and triple spacing (default is single). Margins will be roughly controlled by the character line length selected, either 80 or 132 characters per line (default is 80). Indentation of the CSD constructs has been a constant three blank characters. Support for variable margins and indentation is being investigated in conjunction with the new display routines. In addition, several display options involving CSD graphical constructs are under consideration. For example, the boxes drawn around procedure and task entry calls may be optionally suppressed to make the diagram more compact.

Collapsing the CSD. The CSD window should provide the user with the capability to collapse the CSD based on all control constructs as well as complete diagram entities (e.g., procedures, functions, tasks and packages). This capability directly combines the ideas of chunking with control flow which are major aids to comprehension of software. An *architectural CSD* (ArchCSD) [DAV90] can be facilitated by collapsing the CSD based on procedure, function, and task entry calls, and the control constructs that directly affect these calls. In future versions of the prototype, the ArchCSD will be generated by the display routines from the single intermediate representation of the CSD.

Color. Although general color options such as background and foreground may be selected via the X Windows system, color options within a specific diagram were only briefly investigated for both the screen and printer. It was decided that these will not be pursued in the Version 3 prototype.

Printing An Entire Set of CSDs. Printing an entire set of CSDs in an organized and efficient manner is an important capability when considering the typically large size of Ada software systems. A book format is under consideration which would include a table of contents and/or index. In the event GRASP/Ada is fully integrated with IDE/StP/ADE, the StP Document Preparation System may be utilized for this function.

5.4 Incremental Changes to the CSD

In the present prototype, there is no capability for incrementally modifying the CSD. When the CSD or source code is modified in the CSD window, the CSD must be regenerated. While this has been sufficient for prototyping, especially for small programs, editing capabilities with incremental modification of the CSD are desirable in an operational setting.

5.5 Internal Representation of the CSD - Alternatives

Several alternatives were considered for the internal representation of the CSD in the Version 3 prototype. Each has its own merits with respect to processing and storage efficiency and is briefly described below. These alternatives remain under consideration for future versions.

Single ASCII File with CSD Characters and Text Combined. This is the most direct approach and is currently used in the Version 3 prototype. The primary advantage of this approach is that combining the CSD characters with text in a single file eliminates the need for elaborate transformation and thus enables the rapid implementation of prototypes as was the case in the previous phases of this project. The major disadvantages of this approach are that it does not lend itself to incremental changes during editing and the CSD characters have to be filtered if the user wants to regenerate the CSD or send the file to a compiler.

Separate ASCII Files for CSD Characters and Text. In this approach, the file containing the CSD characters along with placement information would be "merged" with the prettyprinted source file. The primary advantage of the this approach is that the CSD characters would not have to be stripped out if the user wants to send the file to a compiler. The major disadvantage of this approach is that coordinating the two files would add complexity to generation and editing routines with little or no benefit. As a result, this approach would be more difficult to implement than the single file approach and not provide the advantages of the next alternative.

Single ASCII File Without Hard-coded CSD Characters. This approach represents a compromise between the previous two. While it uses a single file, only "begin construct" and "end construct" codes are actually required for each CSD graphical construct in the CSD file rather than all CSD graphics characters that compose the diagram. In particular, no continuation characters would be included in the file. These would be generated by the screen display and print routines as required. The advantages of this approach would be most beneficial in an editing mode since the diagram could grow and shrink automatically as additional text/source code is inserted into the diagram. The extent of required modifications to text edit windows must be considered with this alternative.

Direct Generation From DIANA Net. If tight coupling and integration with a commercial Ada development system such as Verdix VADS is desired, then direct generation of the CSD from the DIANA net produced as a result of compilation could be performed. This would require a layer of software which traverses the DIANA net and calls the appropriate CSD primitives as control nodes are encountered. This approach would eliminate the possibility of directly editing the CSD since the DIANA interface does not support modifying the net, only reading it.

5.6 Navigating among CSDs and Object Diagrams - Future Considerations

A GRASP library is required to provide the overall organization of the generated diagrams. The current file organization uses standard UNIX directory conventions as well as default naming conventions. For example, all Ada source files end in ***.a*** or ***.ada***, the corresponding CSD files end in *.a.csd*, and the corresponding print files end in *.a.csd.ps*. In the present prototype, library complexity has been keep to a minimum by relying on the UNIX directory organization. In future versions, a GRASP library entry will be generated for each procedure, function, package, task, task entry, and label. The library entry will contain minimally the following fields.

identifier - note: unique key should be composed of the identifier + scoping.

scoping/visibility

type (procedure, function, etc.)

parameter list - to aid in overload resolution.

source file (file name, line number) - note: the page number can be computed from the line number.

CSD file (file name, line number)

OD file (file name)

"Referenced by" list

"References to" list

Alternatives for generation and updating of the library entries include the following.

(1) During CSD generation, the library entry is established and the entry is updated on subsequent CSD generations.

(2) During the processing of DIANA nets.

Alternatives for implementing the GRASP library include (1) developing an Ada package or equivalent C module which is called by the CSD generation routines during the parse of the Ada source, (2) using the VADS library system along with DIANA, and (3) using the StP TROLL/USE relational database system. Of these alternatives, the first one may be the most direct approach since it would be the easiest to control. The VADS and StP library approaches may be more useful with the addition of object diagram generation and also with future integration of GRASP with commercial CASE tools.

6.0 Evaluation of the Control Structure Diagram and GRASP/Ada

An important aspect of any research project is the evaluation of the results. In the GRASP/Ada project the two primary results were (1) the development of the Control Structure Diagram (CSD) as a new algorithmic level graphical representation for Ada software and (2) the development of a prototype that automatically generates the CSD from existing Ada PDL or source code. Formal statistically-based controlled experiments dealing with the comprehensibility of graphical representations of software are difficult to design and conduct. Similar difficulties are encountered when attempting design controlled experiments to evaluate CASE tools with respect to improvements in productivity that result from their use. The primary difficulty arises from the learning curve that users/subjects must overcome. For example, a year or more may be required to become proficient enough with a software tool to actually realize gains in productivity. Thus, it may be difficult to compare two CASE tools in a "controlled" experiment without introducing bias based on familiarity or in many cases the lack of familiarity. As a result, most evaluation of CASE tools is based on preference surveys in which the user/subject is asked to make mental assessments or comparisons of various aspects of the tool(s) under study.

This section describes the subjects that participated in the evaluation of the CSD and GRASP/Ada, the preference survey instrument that was developed and administered, and the results of the analysis of the data collected.

6.1 The Subjects

The evaluation instrument was administered to 33 junior/senior computer science and engineering students at Auburn University in the course CSE 422 - Introduction to Software Engineering, during the Fall 1992 quarter. These students all had experience with FORTRAN, Pascal, and C in previous courses. None had formal training in Ada for which the GRASP/Ada tool was designed. Since participation in the evaluation was optional, five bonus points to be added to the final exam score were offered as an incentive. All students present took part in the evaluation.

Each of the graphical representations included in the first part of the evaluation instrument was presented briefly in class, and exercises were assigned involving the Nassi-Shneiderman diagram (NS) and the control structure diagram (CSD). Most students were familiar with the flowchart (FC) from prior classes.

The GRASP/Ada prototype was presented during a laboratory session and used in conjunction with the commercial CASE tool, *Software through Pictures* (StP), which was the primary focus of the CSE 422 lab.

6.2 The Evaluation Instrument

The evaluation instrument was divided into two parts: (1) the evaluation of graphical representations of algorithms and (2) the evaluation of GRASP/Ada (see Appendix C). In the first part, five graphical representations were compared: the ANSI flowchart (FC), the Nassi-Shneiderman diagram (NS), the Warnier-Orr diagram (WO), the action diagram (AD), and the control structure diagram (CSD). The first three items solicited background information with respect to familiarity. In the next eleven items, subjects were asked to compare the diagrams with respect to (a) how well each represented sequence, selection, and iteration, (b) overall readability, (c) improvement in readability as an extension to pseudo-code, (d) ease of coding from, (e) ease of manual use, (f) overall preference if drawn manually, (g) overall economy, (h) overall preference with equivalent automated support, and finally (h) overall preference all assumptions aside. These eleven items are described in more detail blow in the discussion of results. The first part of the instrument concluded with an open ended question soliciting suggestions on how to improve any of the diagrams compared.

The second part of the evaluation instrument was directed specifically at the GRASP/Ada prototype. Questions were designed to solicit information regarding the User Interface, major problems encountered, modifications/enhancements desired, and the level of coverage provided for Ada during the presentation of GRASP/Ada.

6.3 The Evaluation Results

An item analysis was performed on the data collected on the first part of the evaluation instrument with the exception of the three background items at the beginning and the last item which asked for suggested improvements to the diagrams. The results are presented below followed by a general summary of the responses from the second part.

6.3.1 Item Analysis of Comparison of Graphical Representations

An item analysis was performed on eleven items in the first part of the evaluation. The subjects were given the following instructions:

> Based on the experience you have gained by using these diagramming tools to represent algorithms, you are asked to assign a rating to each of the diagrams with respect to a specific comparison among the diagrams. You may assign the same rating to more than one diagram for a given comparison. Select your ratings from the following scale and enter them as indicated below.
>
> 5 - best / most / first choice
> 4 -
> 3 - moderate
> 2 -
> 1 - worst / least / last choice

For each of the eleven items below, the subjects used the rating scale above to complete the following.

FC	NS	WO	AD	CSD
__	__	__	__	__

The eleven items were:

1. Compare the diagrams with respect to how well each shows **sequence.**

2. Compare the diagrams with respect to how well each shows **selection.**

3. Compare the diagrams with respect to how well each shows **iteration.**

4. Compare these diagrams with respect to overall **readability** (consider reading someone else's code).

5. Each of these tools can be used with informal pseudocode as opposed to actual statements in a programming language and, as such, can be thought of as a graphical extension to pseudocode (with possibly some spatial rearrangement). Rate the diagrams on the extent to which they increase readability over non-graphical pseudocode.

6. Suppose as a programmer you are given a design specification in which the program logic has been documented using one of the graphical representations below. Compare the diagrams with respect to which would best facilitate your task of coding from the design specification.

7. Compare the diagrams with respect to **ease of manual use**; consider the initial drawing and subsequent modifications.

8. Assuming you have to manually draw the diagrams (in the sense that they are not automatically generated), indicate your **overall preference** for each diagram where: 5 - first choice, . . ., 1 - last choice.

9. Compare the diagrams with respect to their **overall economy** (i.e., increases in comprehension versus effort to draw them manually).

10. Assuming you have equivalent **automated support** to draw each of the diagrams in the sense that the diagrams are automatically generated either by selecting constructs from a menu or by recognizing key words in the code, indicate your overall preference for each diagram where: 5 - first choice, . . ., 1 - last choice.

11. All assumptions aside, indicate your **overall preference** for each diagram where: 5 - first choice, . . ., 1 - last choice.

The results of the item analysis for these eleven items are contained in the eight tables below. Since their contents is relatively self-explanatory, only a brief interpretation is included after each table.

Table 1. ITEM ANALYSIS FOR GRAPHICAL REPRESENTATIONS

ITEM#	N:items	FC	NS	WO	AD	CSD
1. SEQ	33	3.21	3.64	2.64	2.32	3.94
2. SEL	33	3.52	4.06	2.46	2.05	3.64
3. ITR	33	3.45	3.48	2.58	2.14	3.91
4. GEN READ	33	3.03	3.38	2.67	2.10	4.24
5. EXT P-COD	33	2.85	3.76	2.38	2.48	3.94
6. CODE-FROM	33	2.82	3.53	2.60	2.14	4.31
7. MANUAL	33	3.09	3.16	2.61	2.38	3.91
8. PREF/MANL	33	3.00	3.30	2.42	2.16	4.15
9. ECONOMY	33	2.70	3.27	2.52	2.00	4.52
10. PREF/AUTO	33	3.03	3.52	2.36	2.09	4.55
11. PREF/GEN	33	3.00	3.33	2.54	1.96	4.55
ITEM AVG	363	3.06	3.49	2.52	2.16	4.15

The results in the table above are the averages of the 1 to 5 ratings each of the graphical representations received from the subjects for the eleven items. The results clearly indicate the overall preference for the CSD. Item 5 is of particular interest in that it attempts to determine perceived improvements in readability over non-graphical pseudo-code.

Table 2. ITEM RESPONSE FREQUENCY

ITEM#	N:items	FC	NS	WO	AD	CSD
1. SEQ	33	33	33	25	22	33
2. SEL	33	33	33	24	21	33
3. ITR	33	33	33	24	21	33
4. GEN READ	33	33	32	24	20	33
5. EXT P-COD	33	33	33	24	21	33
6. CODE-FROM	33	33	32	25	22	32
7. MANUAL	33	33	32	23	24	33
8. PREF/MANL	33	33	33	26	25	33
9. ECONOMY	33	33	33	25	24	33
10. PREF/AUTO	33	33	33	25	23	33
11. PREF/GEN	33	33	33	26	25	33

The table above was included to identify those items and graphical representations where subjects left the response blank. A blank indicated the subject was unfamiliar with the notation or particular construct. Averages were computed on the basis of only those items completed.

Table 3. PERCENTAGE SCORE FOR GRAPHICAL REPRESENTATIONS

ITEM#	N:items	FC	NS	WO	AD	CSD
1. SEQ	33	64.24	72.73	52.80	46.36	78.79
2. SEL	33	70.30	81.21	49.17	40.95	72.73
3. ITR	33	69.09	69.70	51.67	42.86	78.18
4. GEN READ	33	60.61	67.50	53.33	42.00	84.85
5. EXT P-COD	33	56.97	75.15	47.50	49.52	78.79
6. CODE-FROM	33	56.36	70.62	52.00	42.73	86.25
7. MANUAL	33	61.82	63.12	52.17	47.50	78.18
8. PREF/MANL	33	60.00	66.06	48.46	43.20	83.03
9. ECONOMY	33	53.94	65.45	50.40	40.00	90.30
10. PREF/AUTO	33	60.61	70.30	47.20	41.74	90.91
11. PREF/GEN	33	60.00	66.67	50.77	39.20	90.91
ITEM AVG	363	61.27	69.89	50.48	43.23	82.98

The table above shows the item averages from Table 1 converted to percentages to provide an additional perspective for comparison. In particular, the differences in percentages among the responses are shown the tables that follow.

Table 4. PERCENTAGE SCORE DIFFERENCE - CSD Compared to Others

ITEM#	N:items	FC	NS	WO	AD	CSD
1. SEQ	33	14.55	6.06	25.99	32.42	
2. SEL	33	2.42	-8.48	23.56	31.77	
3. ITR	33	9.09	8.48	26.52	35.32	
4. GEN READ	33	24.24	17.35	31.52	42.85	
5. EXT P-COD	33	21.82	3.64	31.29	29.26	
6. CODE-FROM	33	29.89	15.62	34.25	43.52	
7. MANUAL	33	16.36	15.06	26.01	30.68	
8. PREF/MANL	33	23.03	16.97	34.57	39.83	
9. ECONOMY	33	36.36	24.85	39.90	50.30	
10. PREF/AUTO	33	30.30	20.61	43.71	49.17	
11. PREF/GEN	33	30.91	24.24	40.14	51.71	
ITEM AVG	363	21.72	13.09	32.50	39.76	

The table above shows the difference between the control structure diagram (CSD) percentage scores and each of the other percentage scores. Positive values indicate preference for the CSD. Note that the NS selection construct was the only item for which the CSD construct was not preferred on average.

Table 5. PERCENTAGE SCORE DIFFERENCE - AD Compared to Others

ITEM#	N:items	FC	NS	WO	AD	CSD
1. SEQ	33	-17.88	-26.36	-6.44		-32.42
2. SEL	33	-29.35	-40.26	-8.21		-31.77
3. ITR	33	-26.23	-26.84	-8.81		-35.32
4. GEN READ	33	-18.61	-25.50	-11.33		-42.85
5. EXT P-COD	33	-7.45	-25.63	2.02		-29.26
6. CODE-FROM	33	-13.64	-27.90	-9.27		-43.52
7. MANUAL	33	-14.32	-15.62	-4.67		-30.68
8. PREF/MANL	33	-16.80	-22.86	-5.26		-39.83
9. ECONOMY	33	-13.94	-25.45	-10.40		-50.30
10. PREF/AUTO	33	-18.87	-28.56	-5.46		-49.17
11. PREF/GEN	33	-20.80	-27.47	-11.57		-51.71
ITEM AVG	363	-18.04	-26.66	-7.25		-39.76

The table above shows the difference between the action diagram (AD) percentage scores and each of the other percentage scores. Negative values indicate a lack of preference for the AD.

Table 6. PERCENTAGE SCORE DIFFERENCE - WO Compared to Others

ITEM#	N:items	FC	NS	WO	AD	CSD
1. SEQ	33	-11.44	-19.93		6.44	-25.99
2. SEL	33	-21.14	-32.05		8.21	-23.56
3. ITR	33	-17.42	-18.03		8.81	-26.52
4. GEN READ	33	-7.27	-14.17		11.33	-31.52
5. EXT P-COD	33	-9.47	-27.65		-2.02	-31.29
6. CODE-FROM	33	-4.36	-18.62		9.27	-34.25
7. MANUAL	33	-9.64	-10.95		4.67	-26.01
8. PREF/MANL	33	-11.54	-17.60		5.26	-34.57
9. ECONOMY	33	-3.54	-15.05		10.40	-39.90
10. PREF/AUTO	33	-13.41	-23.10		5.46	-43.71
11. PREF/GEN	33	-9.23	-15.90		11.57	-40.14
ITEM AVG	363	-10.79	-19.41		7.25	-32.50

The table above shows the difference between the Warnier-Orr (WO) percentage scores and each of the other percentage scores. Positive values indicate preference and negative values indicate a lack of preference for the WO.

Table 7. PERCENTAGE SCORE DIFFERENCE - NS Compared to Others

ITEM#	N:items	FC	NS	WO	AD	CSD
1. SEQ	33	8.48		19.93	26.36	-6.06
2. SEL	33	10.91		32.05	40.26	8.48
3. ITR	33	0.61		18.03	26.84	-8.48
4. GEN READ	33	6.89		14.17	25.50	-17.35
5. EXT P-COD	33	18.18		27.65	25.63	-3.64
6. CODE-FROM	33	14.26		18.62	27.90	-15.62
7. MANUAL	33	1.31		10.95	15.62	-15.06
8. PREF/MANL	33	6.06		17.60	22.86	-16.97
9. ECONOMY	33	11.52		15.05	25.45	-24.85
10. PREF/AUTO	33	9.70		23.10	28.56	-20.61
11. PREF/GEN	33	6.67		15.90	27.47	-24.24
ITEM AVG	363	8.62		19.41	26.66	-13.09

The table above shows the difference between the Nassi-Shneiderman diagram (NS) percentage scores and each of the other percentage scores. Positive values indicate preference for the NS and negative values indicate a lack of preference. Note that the NS selection construct was the only item for which the CSD construct was not preferred.

Table 8. PERCENTAGE SCORE DIFFERENCE - FC Compared to Others

ITEM#	N:items	FC	NS	WO	AD	CSD
1. SEQ	33		-8.48	11.44	17.88	-14.55
2. SEL	33		-10.91	21.14	29.35	-2.42
3. ITR	33		-0.61	17.42	26.23	-9.09
4. GEN READ	33		-6.89	7.27	18.61	-24.24
5. EXT P-COD	33		-18.18	9.47	7.45	-21.82
6. CODE-FROM	33		-14.26	4.36	13.64	-29.89
7. MANUAL	33		-1.31	9.64	14.32	-16.36
8. PREF/MANL	33		-6.06	11.54	16.80	-23.03
9. ECONOMY	33		-11.52	3.54	13.94	-36.36
10. PREF/AUTO	33		-9.70	13.41	18.87	-30.30
11. PREF/GEN	33		-6.67	9.23	20.80	-30.91
ITEM AVG	363		-8.62	10.79	18.04	-21.72

The table above shows the difference between the ANSI Flowchart (FC) percentage scores and each of the other percentage scores. Positive values indicate preference for the FC and negative values indicate a lack of preference. Note that the FC was consistently preferred over the WO and AD. However, the CSD and NS were consistently preferred over the FC.

6.3.2 Summary of Responses For Evaluation of GRASP/Ada

The second part of the evaluation instrument was specifically directed at GRASP/Ada. The items are presented below with a summary of the responses in *italics*.

1. Was the User Interface intuitive?

Most subjects felt comfortable with the User Interface after several sessions. However, many expressed the desire for a User Manual.

2. What changes would you make to the User Interface?

Most subjects stated the User Interface was acceptable as is. Several expressed a desire to have "stickable" subwindows from which options are selected. These were *not available through Athena widgets from which the User Interface was constructed.*

3. What were the major problems you encountered when using GRASP/Ada.

As one might expect, a variety of responses were given for this item. Most were as a result of several known bugs which have since been removed. Some simply indicated improper use of the prototype and/or a lack of expertise in Ada. Again, many expressed the desire for a User Manual.

4. Rank the following items in order of importance in the prototype. Note, some of these items are available in the current version and others are under consideration as modifications/enhancements. Also, feel free to comment on each in the space provided. (1 - least important, ..., 7 - most important)

   The overall rank of the items is indicated.

   a. <u>4.69</u> Integration of CSD generation/editing capabilities with a CASE tool such as StP to facilitate development of process pspecs and/or module PDL.

   b. <u>4.84</u> GRASP/Ada User's Manual.

   c. <u>4.84</u> Error messages resulting from CSD generation.

   d. <u>4.47</u> Integration of CSD editing/generation with automatic generation of object diagrams to show software architectural design (i.e., the object diagrams indicate the dependencies among a set of CSDs).

   e. <u>2.81</u> Spatial options (line spacing, amount of indentation, etc.).

   f. <u>4.22</u> Direct access to a compiler from the User Interface to facilitate use of the CSD during implementation.

   g. <u>5.19</u> Extension of the CSD editor and generator to handle other languages such as C and Pascal.

5. Rate your knowledge of Ada.

_____ excellent _____ good _____ moderate _____ very little _____ virtually none

<u>1.76</u> *indicates knowledge of Ada was between <u>virtually none</u> and <u>very little</u>.*

6. How useful was the Ada template feature in the CSD Window in producing Ada/PDL CSDs?

_____ extremely _____ very _____ moderately _____ not very _____ not useful

<u>3.53</u> *indicates usefulness of the Ada template was between <u>moderately</u> and <u>very useful</u>.*

What modifications/improvements should be made to this feature?

Many subjects indicated that additional Ada construct templates were needed. Only control structures are included presently.

7. The time in class spent on Ada and/or AdaPDL

_____ should have been increased. _____ was about right. _____ should have been decreased.

<u>2.64</u> *indicates the class time spent on Ada was between <u>about right</u> and <u>should have been increased</u>.*

Comments? *Some subjects indicated that the course (CSE 422) should have a more formal emphasis on Ada. Other indicated an emphasis on Pascal or C was preferred since prior required courses cover these languages.*

8. CSD editors and generators are planned for C and Pascal. If these tools were available on the network, how useful would they be to you with respect to improving the readability of your source code in future software development projects?

C:
_____ extremely _____ very _____ moderately _____ not very _____ not useful

<u>4.09</u> *indicates a CSD editor/generator for C would be between <u>very</u> and <u>extremely useful</u>.*

Pascal:
_____ extremely _____ very _____ moderately _____ not very _____ not useful

<u>3.36</u> *indicates a CSD editor/generator for Pascal would be between <u>moderately</u> and <u>very useful</u>.*

7.0 Conclusions and Future Directions

The GRASP/Ada project has provided a strong foundation for the automatic generation of graphical representations from existing Ada software. The current prototype provides the capability for the user to generate the Control Structure Diagram (CSD) from Ada PDL or source code in a reverse engineering mode with a level of flexibility suitable for practical application. The prototype is being used in two software engineering courses at Auburn University on student projects in conjunction with other CASE tools. The feedback provided by the students has been very useful, especially with respect to the user interface.

An important issue for all software tools in general, and graphical representations in particular, is evaluation. The prototype has been prepared for limited distribution (GRASP/Ada Version 3.2) to facilitate evaluation. Although experience indicates that empirical evaluation of graphical notations such as data flow diagrams, object diagrams, structure charts, and flowgraphs is difficult, an evaluation of the CSD for Ada is planned. However, prior to controlled experiments, an informal preference evaluation was conducted to provide preliminary information on actual usage patterns for the CSD. Analysis indicated a clear preference for the CSD over the other graphical notations for algorithms compared.

The CSD generation component of GRASP/Ada has been loosely integrated with IDE's Software though Pictures to replace non-graphical process specifications (pspecs) for data flow diagrams and module PDL for structure charts and object diagrams (see Appendix B). In fact, the CSD becomes a natural detailed-level graphical extension for these system and architectural level diagrams. In this capacity, the CSD has the potential to replace traditional non-graphical pspecs and PDL used in software design and textual source code listings used in implementation, testing, and maintenance.

The primary impact of reverse engineering graphical representations will be improved comprehension of software in the form of visual verification and validation (V & V). To move the results of this research in the direction of visualizations to facilitate the processes of V & V, numerous additional capabilities must be explored and developed. A set of graphical representations that directly support V & V of software at the architectural and system levels of abstraction must be formulated. For example, the Object Diagram generator (ODgen) prototype described earlier is one the components of the GRASP/Ada project which would address architectural and system levels of abstraction. This task must include an on-going investigation of visualizations reported in the literature as currently in use or in the experimental stages of research and development. In particular, specific applications of visualizations to support V & V procedures must be investigated and classified. Prototype software tools which generate visualizations at various levels of abstraction from source code and PDL, as well as other intermediate representations, must be designed and implemented. Graphically-oriented editors must provide capabilities for dynamic reconstruction of the diagrams as changes are made to other diagrams at various levels. These graphical representations should provide immediate visual feedback to the user in an incremental fashion as individual structural and control constructs are completed.

The current prototype of the CSD generator, while only one of set of required visualization tools, has clearly indicated the utility of the CSD. Future enhancements will only increase its effectiveness as a tool for improving the comprehensibility of software.

REFERENCES

ADA83	*The Programming Language Ada Reference Manual.* ANSI/MIL-STD-1815A-1983. (Approved 17 February 1983). In *Lecture Notes in Computer Science*, Vol. 155. (G. Goos and J. Hartmanis, eds) Berlin : Springer-Verlag.
AOY89	M. Aoyama, et.al., "Design Specification in Japan: Tree-Structured Charts," *IEEE Software*, Mar. 1989, 31-37.
BAR84	J. G. P. Barnes, *Programming in Ada*, Second Edition, Addison-Wesley Publishing Co., Menlo Park, CA, 1984.
CHI90	E. J. Chikofsky and J. H. Cross, "Reverse Engineering and Design Recovery - A Taxonomy," *IEEE Software*, Jan. 1990, 13-17.
CRO88	J. H. Cross and S. V. Sheppard, "The Control Structure Diagram: An Automated Graphical Representation For Software," *Proceedings of the 21st Hawaii International Conference on Systems Sciences* (Kailui-Kona, HA, Jan. 5-8). IEEE Computer Society Press, Washington, D. C., 1988, Vol. 2, pp. 446-454.
CRO90a	J. H. Cross, K. I. Morrison, C. H. May, "Generation of Graphical Representations From Source Code," *Proceedings of the Southeast Regional ACM Computer Science Conference*, April 18-20, 1990, Greenville, South Carolina, 54-62.
CRO90b	J. H. Cross, S. V. Sheppard and W. H. Carlisle, "Control Structure Diagrams for Ada," *Journal of Pascal, Ada, and Modula 2*, Vol. 9, No. 5, Sep./Oct. 1990.
CRO92	J. H. Cross, E. J. Chikofsky and C. H. May, "Reverse Engineering," *Advances in Computers*, Vol. 35, 1992, in process.
DAV90	R. A. Davis, "A Reverse Engineering Architectural Level Control Structure Diagram," *M.S. Thesis*, Auburn University, December 14, 1990.
MAR85	J. Martin and C. McClure, *Diagramming Techniques for Analysts and Programmers*. Englewood Cliffs, NJ : Prentice-Hall, 1985.
SCA89	D. A. Scanlan, "Structured Flowcharts Outperform Pseudocode: An Experimental Comparison," IEEE Software, Sep. 1989, 28-36.
SEL85	R. Selby, et. al., "A Comparison of Software Verification Techniques," NASA *Software Engineering Laboratory Series* (SEL-85-001), Goddard Space Flight Center, Greenbelt, Maryland, 1985.

SHN77	B. Shneiderman, et. al., "Experimental Investigations of the Utility of Detailed Flowcharts in Programming," *Communications of the ACM*, No. 20 (1977), pp. 373-381.
SHU88	Nan C. Shu, *Visual Programming*, New York, NY, Van Norstrand Reinhold Company, Inc., 1988.
STA85	T. Standish, "An Essay on Software Reuse," *IEEE Transactions on Software Engineering*, SE-10 (9), 494-497, 1985.
TRI89	L. L. Tripp, "A Survey of Graphical Notations for Program Design -An Update," *ACM Software Engineering Notes*, Vol. 13, No. 4, 1989, 39-44.
WAS89	A. I. Wasserman, P. A. Pircher and R. J. Muller, "An Object Oriented Structured Design Method for Code Generation," ACM SIGSOFT *Software Engineering Notes*, Vol. 14, No. 1, January 1989, 32-52.

APPENDICES

Appendix A

GRASP/Ada

Getting Started

Getting Started

Getting Started assumes GR*ASP/Ada* has been properly installed on your local UNIX system. If this has not been done see the README.INSTALL file included with the GRASP/Ada software. The steps below describe required modifications to your .cshrc file and the command that executes GRASP/Ada.

1. An environment variable called GRASP_HOME (where the current version of GRASP is installed) should be set. Contact the system administrator about the GRASP_HOME directory details.

   setenv GRASP_HOME ____________________

2. The GRASP executable is located in $GRASP_HOME/bin directory. So $GRASP_HOME/bin directory should be added to the path list. If path variable has already been set, add the following line after setting the GRASP_HOME environment variable.

   set path = ($path $GRASP_HOME/bin)

3. GRASP man pages are located in $GRASP_HOME/man. So $GRASP_HOME/man should be added to the MANPATH environment variable. If MANPATH environment variable has already been set, add the following line after setting the GRASP_HOME environment variable.

   setenv MANPATH ${MANPATH}:$GRASP_HOME/man

4. Save the .cshrc file and type the following at the command prompt.

   source .cshrc

5. Type ***graspada &*** at the command prompt to execute GRASP in the background.

Appendix B

GRASP/Ada

Integrating GRASP/Ada with *Software through Pictures* (StP)

Integrating GRASP/Ada with *Software through Pictures* (StP)

Introduction

Software through Pictures (StP) is a commercially available CASE tool from Interactive Development Environments, Inc. (IDE). StP provides automated support for software development methods which allow the user to build a comprehensive model of the system. This system model helps ensures the integrity of the design before starting the production of the code. In particular, StP can be used to build the graphical representations such as data flow diagrams, structure charts, entity relationship diagrams, and object diagrams. The information generated from the design is compiled into the underlying database called Data Dictionary to ensure consistency across the entire project. Once the design is complete, the system can be developed according to the model. StP includes a set of editors for graphically modeling programs and the data used in the programs. Among these graphical editors are the Data Flow Diagram Editor (DFE) and Structure Chart Editor (SCE). GRASP/Ada can be used to generate a control structure diagram (CSD) from (or in place of) process specifications (pspecs) in the DFE and module specifications (PDL) in the SCE. Using CSDs for pspecs and PDL provides a natural graphical extension to data flow diagrams and structure charts. The figure below shows a snapshot of the screen with a pspec represented by a CSD.

Data Flow Diagram Editor

The Data Flow Diagram Editor (DFE) is an interactive graphical tool for drawing data flow diagrams in support of the Structured Analysis method. Data flow diagrams provide a view of the system from a functional perspective. The top level context diagram shows the system's overall purpose and how it interacts with external objects. Lower level diagrams show the system subdivided into components or processes using decomposition techniques. The DFE models the data flow of a system by using symbols that represent processes, data flows, data stores, and external data sources and sinks. The status of a process is represented by the presence or absence of a status marker next to the index number. Ifs is undefined. An undefined process is one that is neither decomposed nor has a pspec. All processes must be defined before the diagram is entered into the Data Dictionary. If there is an asterisk (*) next to the process index number, the process is decomposed. If a Pspec has been generated for the process, a small p appears next to the process index.

GRASP/Ada can be used to generate CSDs from syntactically correct Ada programs or Ada PDL. Since the Pspec editor of the DFE does not have the graphic capability, it can be replaced by GRASP/Ada.

Structure Chart Editor

The Structure Chart Editor (SCE) is an interactive graphical editor for drawing structure charts. The purpose of the structure chart is to show the interconnections between identifiable program modules by graphically representing the modules hierarchy and indicating the data that is passed between the modules. Each module has an associated non-graphical PDL module specification. As with the pspec above, GRASP/Ada can be used to generate a CSD for each PDL spec by replacing the StP PDL editor with GRASP/Ada.

Integration Procedure

Before integrating GRASP/Ada with StP, the environment variables and path variables required to execute GRASP/Ada as a stand alone tool should be set in .cshrc file. Tool Information files are the principal means by which the user customizes StP to suit the needs of individual user and environment.

Variables in the Tool Information file are used to customize the environment in which StP runs, specify the commands StP executes when running, and create the look and feel of the graphical editors. The Tool Information file is a ASCII text file, and the variables can be changed by commenting them out, changing their values, or setting their values in another file.

When StP is first invoked from the command line, the system uses routines in the Tools Library to find the appropriate Tool Information file to read. First, the system looks for an environment variable called **ToolInfo** defined in the user's .cshrc file. This variable gives the complete pathname to the Tool Information file to be read. A ToolInfo variable can have a value which is a number or a character string, the pathname for a file or directory, or a command with optional parameters. For example, the variable that specifies the location of the file used to set up the Main Menu may appear in the file as follows:

STPMenuSpec=/usr/local/lib/STPmenu.spec

The appearance of the Main Menu and the tools and commands available through it are controlled by certain ToolInfo variables and Main Menu Specification file.

The Main Menu Specification File. The main menu specification file determines choices available in various areas of the Main Menu Window. It specifies the icons and labels that can appear in these areas, and it determines what combinations of commands or list of choices are available and how they are displayed.

Structure of the Specification File. The following excerpt from STPmenu.spec gives the specification for CSD icon.

```
/GRASP
        label  [GRASP]
        image    { \
/* Format_version=1, Width=64, Height=64, Depth=1, Valid_bits_per_item=16   \
*/    \
    0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF, \
    0xC000,0x0000,0x0000,0x0003,0xC000,0x0000,0x0000,0x0003, \
    0xC000,0x0FFF,0xFFFF,0xFC03,0xC000,0x0FFF,0xFFFF,0xFC03, \
    0xC000,0x0C00,0x0000,0x0003,0xC000,0x0C00,0x0000,0x0003, \
    0xC000,0x0C00,0x0000,0x0003,0xC000,0x0FFF,0xFFFF,0xFC03, \
    0xC000,0x0FFF,0xFFFF,0xFC03,0xC000,0x0001,0x8000,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC000,0x0001,0x8000,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC3E0,0x0001,0x8000,0x0003, \
    0xC7F0,0x0001,0x8060,0x0003,0xCC30,0x0001,0x80F0,0x0003, \
    0xCC01,0xE001,0xFF80,0x0003,0xCC03,0xF001,0xFF98,0x0003, \
    0xCC07,0x3801,0x8198,0x0003,0xCC07,0x1BE1,0x819F,0xE003, \
    0xCC03,0x83F1,0x819F,0xE003,0xCC01,0xC339,0x8198,0x0003, \
    0xCC30,0xE319,0x8198,0x0003,0xC7F0,0x7319,0x819F,0xE003, \
    0xC3E6,0x3B19,0x819F,0xE003,0xC007,0x3B19,0x8198,0x0003, \
    0xC003,0xF319,0x8198,0x0003,0xC001,0xE319,0x819F,0xE003, \
    0xC000,0x0339,0x819F,0xE003,0xC000,0x03F1,0x8198,0x0003, \
    0xC000,0x03E1,0x80F0,0x0003,0xC000,0x0001,0x8060,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC000,0x0001,0x8000,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC000,0x0001,0x8000,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC000,0x0001,0x8000,0x0003, \
    0xC000,0x0001,0x8000,0x0003,0xC1F8,0x0E01,0xFFE0,0x0003, \
    0xC3FC,0x1E01,0xFFE0,0x0003,0xC606,0x1601,0x8000,0x0003, \
    0xC606,0x0601,0x8000,0x0003,0xC006,0x0601,0x8000,0x0003, \
    0xC006,0x0601,0x8000,0x0003,0xC006,0x0601,0x8000,0x0003, \
    0xC006,0x0601,0x8000,0x0003,0xC07C,0x0601,0x8000,0x0003, \
    0xC07C,0x0601,0x8000,0x0003,0xC006,0x0601,0x8000,0x0003, \
    0xC006,0x0601,0x8000,0x0003,0xC006,0x0601,0x8000,0x0003, \
    0xC006,0xC601,0x8000,0x0003,0xC606,0xC601,0x8000,0x0003, \
    0xC606,0x0601,0x8000,0x0003,0xC3FC,0x1F81,0x8000,0x0003, \
    0xC1F8,0x1F81,0x8000,0x0003,0xC000,0x0000,0x0000,0x0003, \
    0xC000,0x0000,0x0000,0x0003,0xC000,0x0000,0x0000,0x0003, \
    0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF  \
}
      help  Graphical Representation of Algorithms, Structures and Processes
      row   0
      col   40

//Edit_Diagrams
      label       [Edit Diagrams]
      cmd         graspada ${Diagram_Name}
      msg         Control Structure Diagram Editor

///Diagram_Name
      label Diagram Name(s):
      text        (,,)
```

All the Hex numbers are the bitmap representation of the CSD icon. The line **cmd graspada ${Diagram_Name}** will specify what command to execute when the user selects CSD icon and clicks on execute.

The **dfe_pspec_edit** variable in ToolInfo file should be set as follows so that it invokes GRASP/Ada instead of standard Pspec editor.

dfe_pspec_edit=graspada&

The **sce_pdl_edit** variable in ToolInfo file should be set as follows so that it invokes GRASP/Ada instead of standard PDL editor.

sce_pdl_edit=graspada&

SUMMARY OF INTEGRATION STEPS

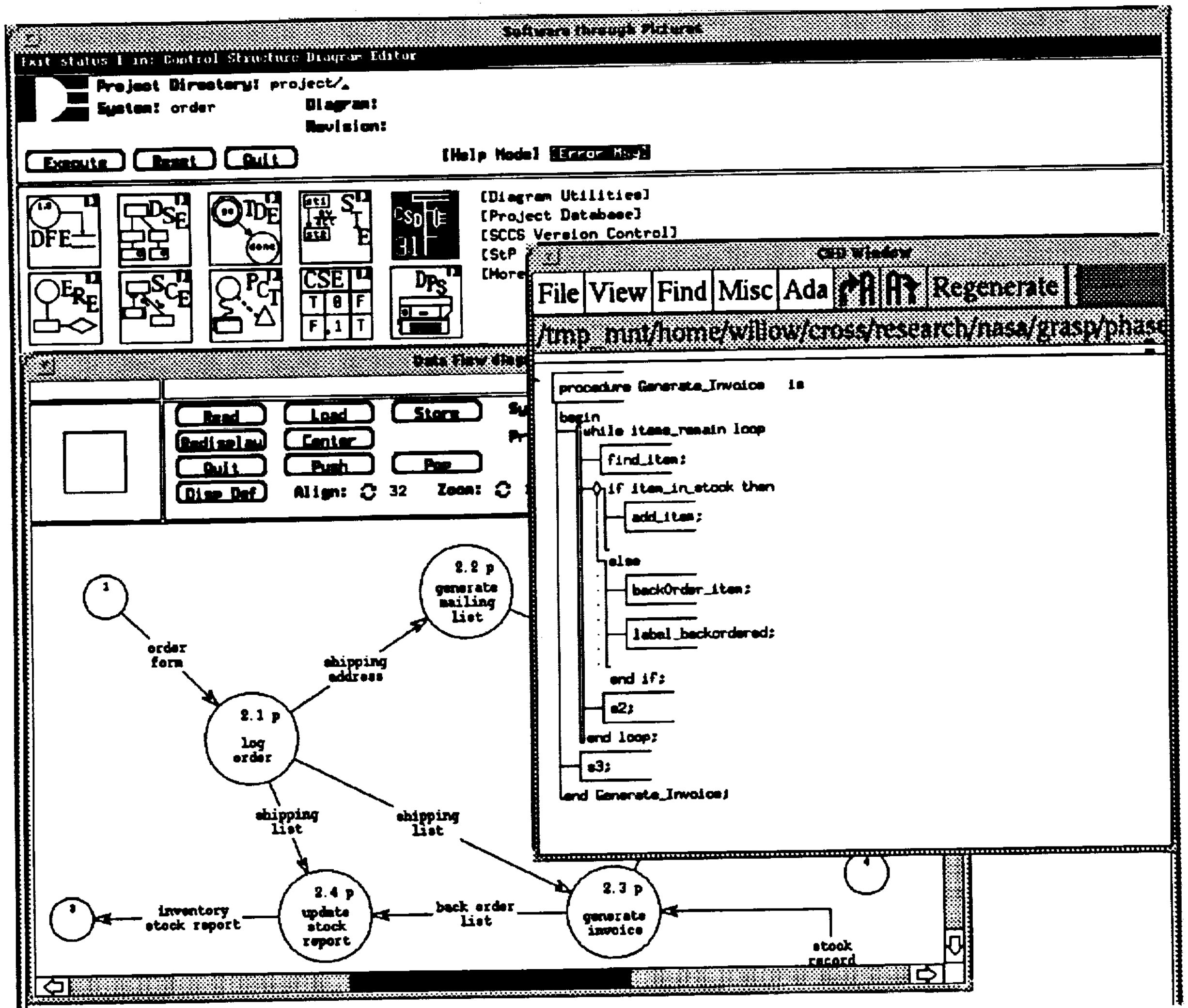
1. Copy the ToolInfo and STPmenu.spec files from StP library to user's home directory. Set the **ToolInfo** environment variable in .cshrc file to refer to the ToolInfo file in user's home directory.

2. Load the ToolInfo file (which is copied into user's home directory) into an editor and modify the ToolInfo variable **STPMenuSpec** as follows:
   STPMenuSpec=~/STPmenu.spec

3. To invoke GRASP/Ada in place of Pspec editor, replace the ToolInfo variable **dfe_pspec_edit** variable as follows:
   dfe_pspec_edit=graspada

4. To invoke GRASP/Ada in place of PDL editor,replace the ToolInfo variable **sce_pdl_edit** variable as follows:
   sce_pdl_edit=graspada

5. To invoke GRASP/Ada as an independent application (like DFE, SCE, etc.,) copy the information provided above in the **Structure of the Specification File** to STPmenu.spec file in user's home directory.



Exit status 1 in: Control Structure Diagram Editor

Project Directory: project/.
System: order
Diagram:
Revision:

/tmp_mnt/home/willow/cross/research/nasa/grasp/phase

```
procedure Generate_Invoice  is
begin
  while items_remain loop
    find_item;
    if item_in_stock then
      add_item;
    else
      backOrder_item;
      label_backordered;
    end if;
    s2;
  end loop;
  s3;
end Generate_Invoice;
```

Figure 14. GRASP/Ada CSD Window with Pspec for Generate_Invoice in StP.

Appendix C

GRASP/Ada

Evaluation Instrument

Name ______________________________ Date ______________________
(optional)

Evaluation of Graphical Representations for Algorithms

Several of the following graphical representations were briefly presented in class: flowcharts (FC), Nassi-Shneiderman diagrams (NS), Warnier-Orr diagrams (WO), action diagrams (AD), and control structure diagrams (CSD).

During this course, which of the above diagrams were presented? Check the appropriate responses.

FC	NS	WO	AD	CSD
__	__	__	__	__

Prior to this course, which of these diagrams had you used? Check the appropriate responses.

FC	NS	WO	AD	CSD
__	__	__	__	__

Were any of the diagrams used in a professional setting? Check the appropriate responses.

FC	NS	WO	AD	CSD
__	__	__	__	__

Based on the experience you have gained by using these diagramming tools to represent algorithms, you are asked to assign a rating to each of the diagrams with respect to a specific comparison among the diagrams. You may assign the same rating to more than one diagram for a given comparison. Select your ratings from the following scale and enter them as indicated below.

5 - best / most / first choice
4 -
3 - moderate
2 -
1 - worst / least / last choice

1. Compare the diagrams with respect to how well each shows **sequence.**

FC	NS	WO	AD	CSD
___	___	___	___	___

2. Compare the diagrams with respect to how well each shows **selection.**

FC	NS	WO	AD	CSD
___	___	___	___	___

3. Compare the diagrams with respect to how well each shows **iteration.**

FC	NS	WO	AD	CSD
___	___	___	___	___

4. Compare these diagrams with respect to overall **readability** (consider reading someone else's code).

FC	NS	WO	AD	CSD
___	___	___	___	___

5. Each of these tools can be used with informal pseudocode as opposed to actual statements in a programming language and, as such, can be thought of as a graphical extension to pseudocode (with possibly some spatial rearrangement). Rate the diagrams on the extent to which they increase readability over non-graphical pseudocode.

FC	NS	WO	AD	CSD
___	___	___	___	___

6. Suppose as a programmer you are given a design specification in which the program logic has been documented using one of the graphical representations below. Compare the diagrams with respect to which would best facilitate your task of coding from the design specification.

FC	NS	WO	AD	CSD
___	___	___	___	___

7. Compare the diagrams with respect to **ease of manual use**; consider the initial drawing and subsequent modifications.

FC	NS	WO	AD	CSD
___	___	___	___	___

8. Assuming you have to manually draw the diagrams (in the sense that they are not automatically generated), indicate your **overall preference** for each diagram where:
5 - first choice, . . ., 1 - last choice

FC	NS	WO	AD	CSD
___	___	___	___	___

9. Compare the diagrams with respect to their **overall economy** (i.e., increases in comprehension versus effort to draw them manually).

FC	NS	WO	AD	CSD
___	___	___	___	___

10. Assuming you have equivalent **automated support** to draw each of the diagrams in the sense that the diagrams are automatically generated either by selecting constructs from a menu or by recognizing key words in the code, indicate your overall preference for each diagram where:
5 - first choice, . . ., 1 - last choice

FC	NS	WO	AD	CSD
___	___	___	___	___

11. All assumptions aside, indicate your **overall preference** for each diagram where:
5 - first choice, . . ., 1 - last choice

FC	NS	WO	AD	CSD
___	___	___	___	___

12. It is not uncommon for individuals and organizations to introduce modifications (which they consider to be improvements) to "standard" diagramming tools. These changes may be to improve readability, to make the diagrams easier to work with manually, to make them easier to automate, to provide for control flow other than sequence, selection, iteration, etc. What **improvements** can you suggest for any of the diagrams we used in this class?

Name ____________________________ Date ______________________
(optional)

Evaluation of GRASP/Ada

In CSE 422 lab you were provided the opportunity to work briefly with GRASP/Ada, a prototype reverse engineering tool for software written in Ada or AdaPDL. The prototype is currently being evaluated prior to widespread release via the network. As a prototype, GRASP/Ada is expected to undergo continual modification over the next year, especially with respect to integration with commercially available CASE tools.

The current GRASP/Ada prototype includes automatic generation of Control Structure Diagrams (CSDs). Future releases will include the generation of object diagrams. Your responses to the items below are intended to provide the developers with directions for enhancements to the prototype, including additional user interface requirements and overall functionality.

The GRASP/Ada Project is supported, in part, by funding from George C. Marshall Space Flight Center, NASA, Alabama 35821 (Contract Number NASA-NCC8-14).

1. Was the User Interface intuitive?

2. What changes would you make to the User Interface?

3. What were the major problems you encountered when using GRASP/Ada.

4. Rank the following items in order of importance in the prototype. Note, some of these items are available in the current version and others are under consideration as modifications/enhancements. Also, feel free to comment on each in the space provided. (1 - least important, ..., 7 - most important)

a. _____ Integration of CSD generation/editing capabilities with a CASE tool such as StP to facilitate development of process pspecs and/or module PDL.

b. _____ GRASP/Ada User's Manual.

c. _____ Error messages resulting from CSD generation.

d. _____ Integration of CSD editing/generation with automatic generation of object diagrams to show software architectural design (i.e., the object diagrams indicate the dependencies among a set of CSDs).

e. _____ Spatial options (line spacing, amount of indentation, etc.).

f. _____ Direct access to a compiler from the User Interface to facilitate use of the CSD during implementation.

g. _____ Extension of the CSD editor and generator to handle other languages such as C and Pascal.

5. Rate your knowledge of Ada.

_____ excellent _____ good _____ moderate _____ very little _____ virtually none

6. How useful was the Ada template feature in the CSD Window in producing Ada/PDL CSDs?

_____ extremely _____ very _____ moderately _____ not very _____ not useful

What modifications/improvements should be made to this feature?

7. The time in class spent on Ada and/or AdaPDL

_____ should have been increased. _____ was about right. _____ should have been decreased.

Comments?

8. CSD editors and generators are planned for C and Pascal. If these tools were available on the network, how useful would they be to you with respect to improving the readability of your source code in future software development projects?

C:

_____ extremely _____ very _____ moderately _____ not very _____ not useful

Pascal:

_____ extremely _____ very _____ moderately _____ not very _____ not useful