A Visualization and Measurement Environment for Software Engineering

T. Dean Hendrix, James H. Cross II, Larry A. Barowski, Joseph C. Teate,
Karl S. Mathias, and Tahia I. Morris
Auburn University
Email: hendrix@eng.auburn.edu

Abstract– Work is reported on the development and enhancement of the GRASP software engineering and visualization tool. GRASP automatically produces visualizations of control and complexity (control structure diagrams and complexity profile graphs, respectively) of source code written in Ada 95, C, and Java. These visualizations use intuitive, compact graphical representations that allow the software engineer to holistically visualize the overall program as well as visualize the details of a small section of the code. Current features and enhancements of GRASP as well as its availability are discussed.

1. Introduction

Software visualization is an active area of research that investigates efficient and effective ways of automatically producing graphical representations of program source code, algorithms, or the runtime behavior of software. Such visualization technology promises to bring considerable help to bear on difficult and costly issues in software engineering such as communicating program information for the purposes of testing, maintenance, and reverse engineering [2].

As part of the ongoing GRASP project at Auburn University, we have developed a software engineering tool, GRASP/Ada that automatically produces a control structure diagram (CSD) visualization of Ada source code and PDL in a manner flexible and efficient enough for professional application [3]. Initial empirical studies indicate the usefulness of the CSD and the practical potential for tools such as GRASP/Ada [4].

Despite the potential usefulness of visualizations such as the CSD, there is considerable evidence that the effectiveness of visualizations is quite sensitive to issues such as the user's skill level (the ability of a user to interpret a given visualization) and the manner in which the visualization is produced (the suitability of a given visualization to be correctly interpreted) [7]. Recognizing these facts, we have enhanced the GRASP software visualization tool to incorporate significant new features which make the resulting visualizations more useful.

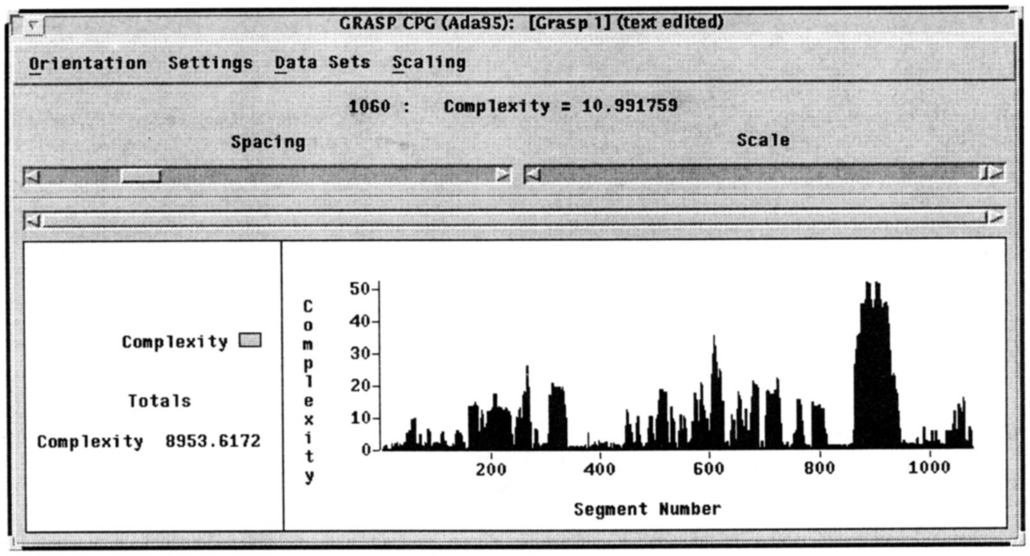
In this paper we discuss three enhancements to GRASP: the measurement of source code through the complexity profile graph, the scalability of visualization through unit symbols, and the extension of GRASP to multiple languages.

2. The Complexity Profile Graph

The complexity profile graph (CPG) [6] is a new fine-grained complexity metric. Unlike other many other complexity metrics, the CPG provides a composite profile of a program's complexity at the individual statement level rather than producing a single metric for the entire program. This composite profile can be viewed as a graph and synchronized with a visualization of source code structure such as the CSD. The statement-level profile nature of the CPG is particularly useful for identifying the complex clusters of code in large programs. The CPG in Figure 1 easily allows the user to quickly identify the complex clusters of code and appropriately concentrate their efforts. This CPG was automatically produced by GRASP and is synchronized with a CSD window (not shown) containing the corresponding source code.

Figure 2 shows a CPG window synchronized with a separate window containing the CSD visualization of the corresponding source code. Once the complex clusters are identified, the user can simply click on a line in the cluster and be automatically scrolled there in the CSD window.

The CPG is composed of five individual measures of statement-level complexity: content complexity, inherent complexity, statement reachability, statement breadth complexity, and total complexity (a weighted sum of the other four). Each of these five measures may be plotted individually, in combination with each other (as shown in Figure 3 and Figure 4) or as a weighted sum called the Total Complexity (as shown in Figure 1 and Figure 2).



```
1060 :   Complexity = 10.991759
```

```
Totals
Complexity  8953.6172
```

Figure 1. CPG Identifying complex clusters of code.

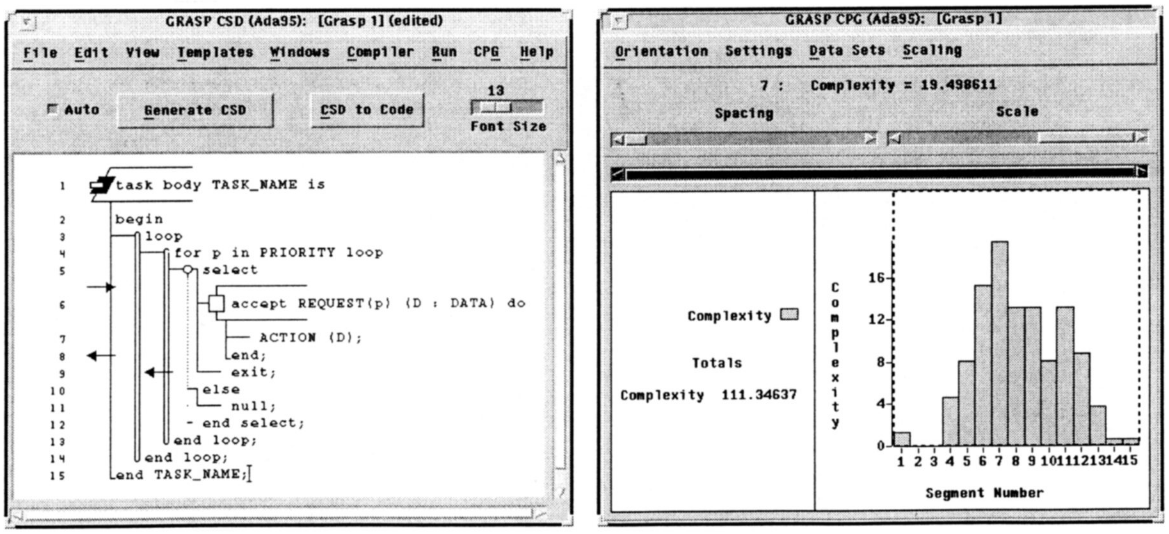
3. CSD Unit Symbols

One of the ways in which visualization tools should aid the software engineer is in scaling from one level of abstraction to another. For example, a reverse engineering effort might begin at the source code level, then progress to the architecture and on to the design level. To provide cognitive leverage to the user in making these transitions, GRASP has added new unit symbols to its notation for program units such as procedures, functions, packages, and tasks.

The standard CSD has always provided the box symbols shown in Figure 5. Now, the user can choose four different modes of viewing program units: box symbols only, unit symbols only, box symbols and unit symbols together, or no symbol at all. The new unit symbols, shown in Figure 6 are patterned after Booch's module notation [1] but include additional original symbols for task entry, protected specification and body, and exception handler. As programs increase in size and complexity, the CSD unit symbols become more useful in comprehending the Ada source code since they can provide a direct visual relation to the architectural diagrams of the system

4. The GRASP Environment

In addition to the visualizations, GRASP provides the user with a complete program development environment, including a full-featured program editor, compiler,

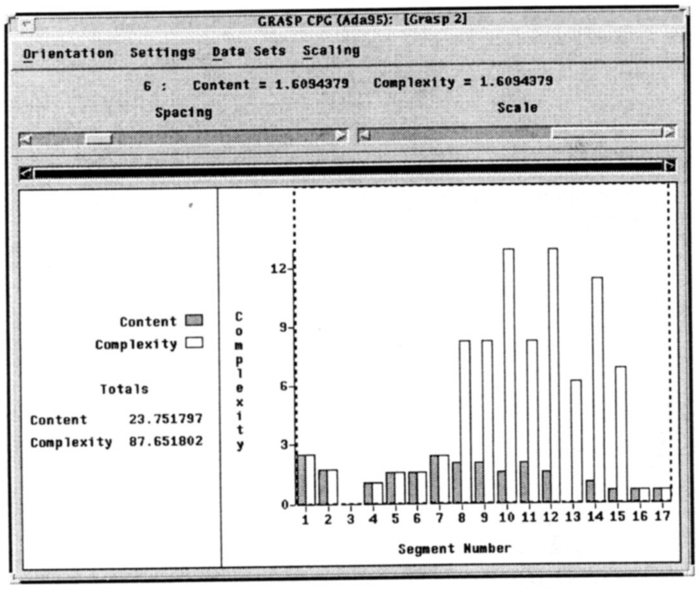


```
task body TASK_NAME is
begin
   loop
      for p in PRIORITY loop
         select
            accept REQUEST(p) (D : DATA) do
               ACTION (D);
            end;
            exit;
         else
            null;
         end select;
      end loop;
   end loop;
end TASK_NAME;
```

```
7 :   Complexity = 19.498611
```

```
Totals
Complexity  111.34637
```

Figure 2. CSD and CPG windows synchronized.


GRASP CPG (Ada95): [Grasp 2]
Orientation Settings Data Sets Scaling
Spacing
Scale
Content
Complexity
Complexity
Segment Number

```
6 :   Content = 1.6094379   Complexity = 1.6094379

Totals
Content       23.751797
Complexity    87.651802
```

Figure 3. CPG plotting Total Complexity and Content Complexity separately.

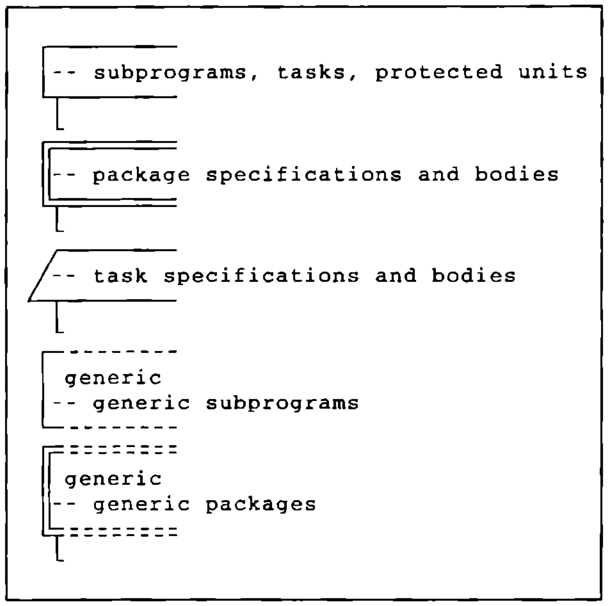
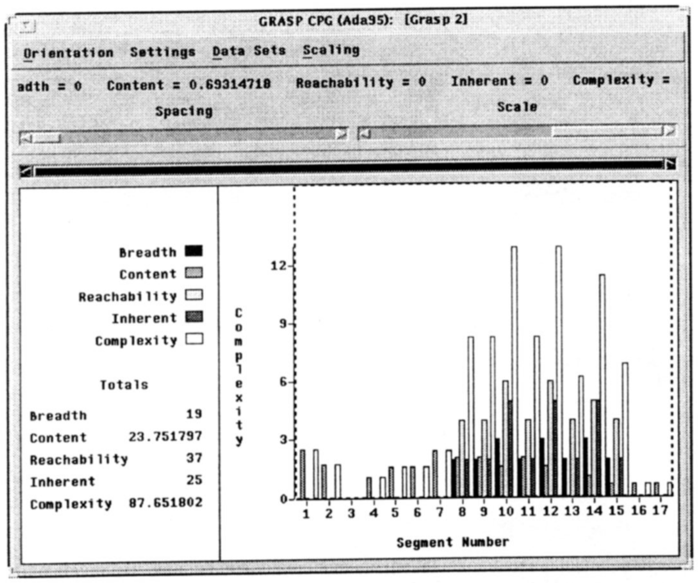

-- subprograms, tasks, protected units
-- package specifications and bodies
-- task specifications and bodies
generic
-- generic subprograms
generic
-- generic packages

Figure 5. CSD Box symbols.



```
adth = 0   Content = 0.69314718   Reachability = 0   Inherent = 0   Complexity =

Totals
Breadth          19
Content   23.751797
Reachability     37
Inherent         25
Complexity 87.651802
```

Figure 4. All five complexity measures plotted separately.

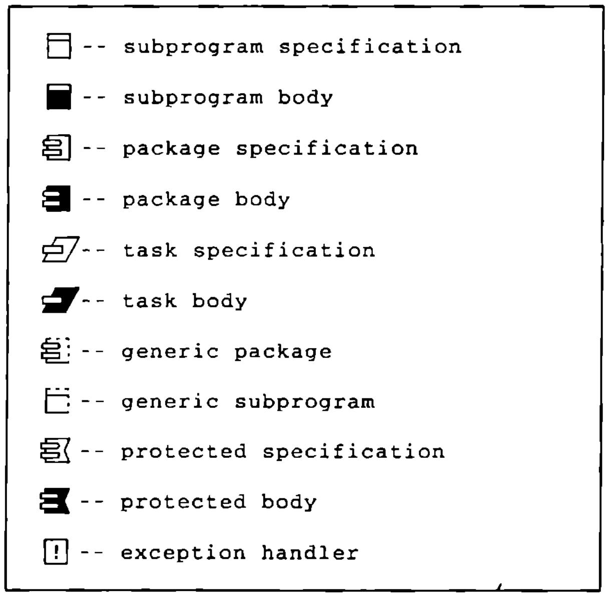


Figure 6. CSD Unit Symbols.

linker, and runtime window. Users organize and control a session with GRASP through the Control Panel, shown in Figure 7. Menu options on the Control Panel provide functions such as selecting of a printer, accessing the user manual, and setting user preferences.

The **File** option allows the user to open one or more CSD windows. Each CSD window, as shown in Figure 8, is a full-function text editor with the additional capabilities to generate, display, edit, and print CSDs of source code and also perform syntax checks, compilations, makes, and executions of the source code. The **File** and **Edit** options are similar to traditional text editors, complete with cut-copy-paste and undo options.

Multiple CSD windows may be opened to access several files at once. File names and their associated directory paths are selected under the **File** option and displayed at the top of each window. Each time a source code file is loaded, the CSD is automatically generated if

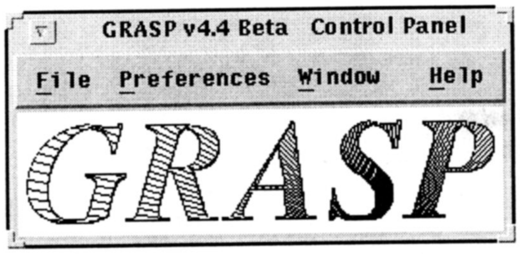
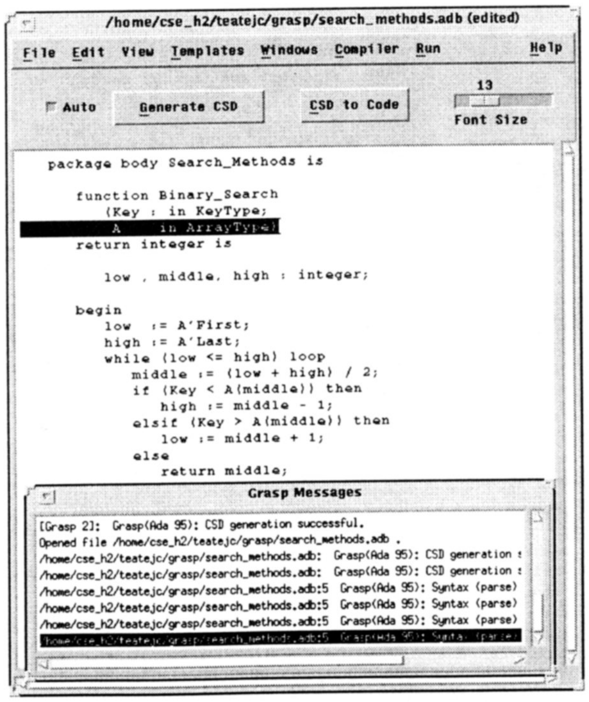


Figure 7. GRASP Control Panel

Auto (next to **Generate CSD** on the toolbar) is turned on. If **Auto** is not turned on, the user may elect to use GRASP only as a text editor and not take advantage of the visualization options, or generate the CSD and CPG visualizations independently and on demand.

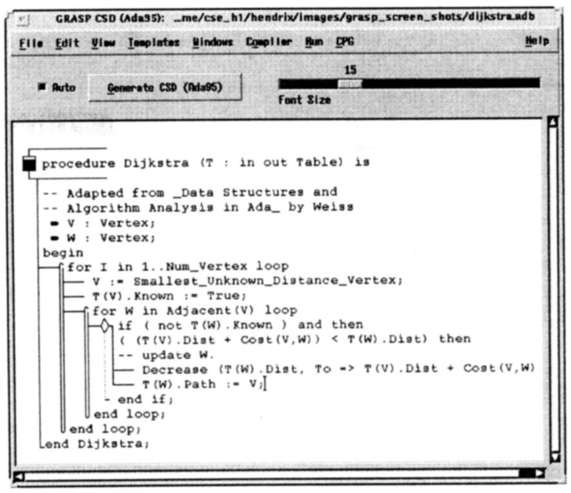
Pressing **Generate CSD** on the toolbar (or control-g, or F1) will immediately render the CSD of the source code in the CSD window. All white space and comments are preserved with the exception of leading indentation with is replaced by the CSD. If a parse error occurs during CSD generation, the cursor is moved to the highlighted line containing the error and a message window appears informing the user of the error, as shown in Figure 9. The CSD generation and display cycle is extremely fast, rendering approximately 20,000 line of code per second on a Sun UltraSparc. Thus, generating the CSD is also a quick way to check for syntax errors without performing a much slower



```
package body Search_Methods is

   function Binary_Search
      (Key : in KeyType;
       A     in ArrayType)
   return integer is

      low , middle, high : integer;

   begin
      low  := A'First;
      high := A'Last;
      while (low <= high) loop
         middle := (low + high) / 2;
         if (Key < A(middle)) then
            high := middle - 1;
         elsif (Key > A(middle)) then
            low := middle + 1;
         else
            return middle;
```

```
[Grasp 2]:  Grasp(Ada 95): CSD generation successful.
Opened file /home/cse_h2/teatejc/grasp/search_methods.adb .
/home/cse_h2/teatejc/grasp/search_methods.adb:  Grasp(Ada 95): CSD generation :
/home/cse_h2/teatejc/grasp/search_methods.adb:  Grasp(Ada 95): CSD generation :
/home/cse_h2/teatejc/grasp/search_methods.adb:5  Grasp(Ada 95): Syntax (parse)
/home/cse_h2/teatejc/grasp/search_methods.adb:5  Grasp(Ada 95): Syntax (parse)
/home/cse_h2/teatejc/grasp/search_methods.adb:5  Grasp(Ada 95): Syntax (parse)
/home/cse_h2/teatejc/grasp/search_methods.adb:5  Grasp(Ada 95): Syntax (parse)
```

Figure 9. GRASP reporting a syntax error.



```
procedure Dijkstra (T : in out Table) is
-- Adapted from _Data Structures and
-- Algorithm Analysis in Ada_ by Weiss
 V : Vertex;
 W : Vertex;
begin
   for I in 1..Num_Vertex loop
      V := Smallest_Unknown_Distance_Vertex;
      T(V).Known := True;
      for W in Adjacent(V) loop
         if ( not T(W).Known ) and then
         ( (T(V).Dist + Cost(V,W)) < T(W).Dist) then
         -- update W.
            Decrease (T(W).Dist, To => T(V).Dist + Cost(V,W)
            T(W).Path := V;
         end if;
      end loop;
   end loop;
end Dijkstra;
```

Figure 8. GRASP CSD Window

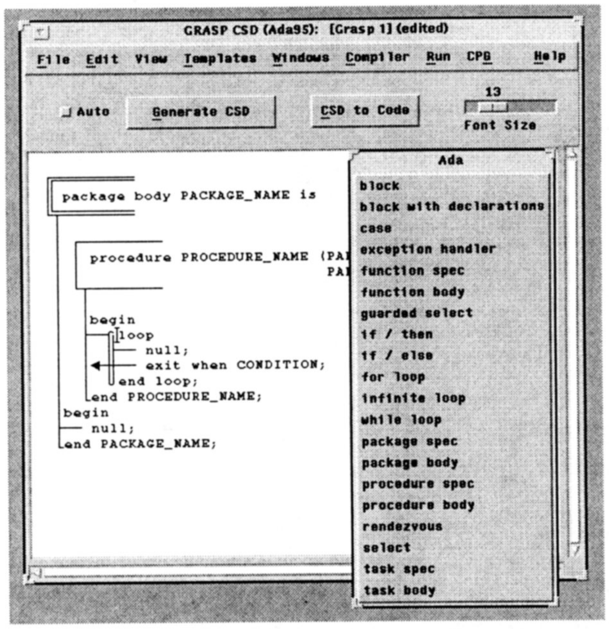
compilation. The net result is that the CSD window can be used in place of a traditional program editor to generate, display, edit, and print CSDs and check for syntax errors with virtually no overhead; i.e., the CSD is essentially free.

View allows the user to select any combination (or none) of the following: the standard CSD box notation, alternate CSD program unit symbols, and line numbers. The **Template** option opens a tear-off menu of templates to aid in writing code. A selection of one of the template choices will cause a block of text, typically a pre-defined program structure, to be pasted into the source code at the point of the cursor. Users can also add any number of personal template items to the list. The CSD in Figure 10 is the result of choosing the Ada program templates "package body", "procedure body", and "infinite loop."

5. Many Languages, One Development Environment

GRASP is designed around the language independent model of rendering described in [5]. Thus, the rendering algorithm is independent of a particular source language, although a certain amount of preprocessing is necessarily language dependent. In the current GRASP prototype, the CSD window is capable of providing the same functionality for many different languages, including Ada, C, and Java. Languages to be included in the near future include C++ and VHDL. Figure 11 and Figure 12 show examples of C and Java programs respectively constructed automatically using GRASP's template feature for these languages.

GRASP users are presented with a single tool and interface for multiple languages, making GRASP well suited to the needs of those involved with the



```
package body PACKAGE_NAME is
  procedure PROCEDURE_NAME (PA
                           PA
  begin
    loop
      null;
      exit when CONDITION;
    end loop;
  end PROCEDURE_NAME;
begin
  null;
end PACKAGE_NAME;
```

Figure 10. GRASP's template feature.

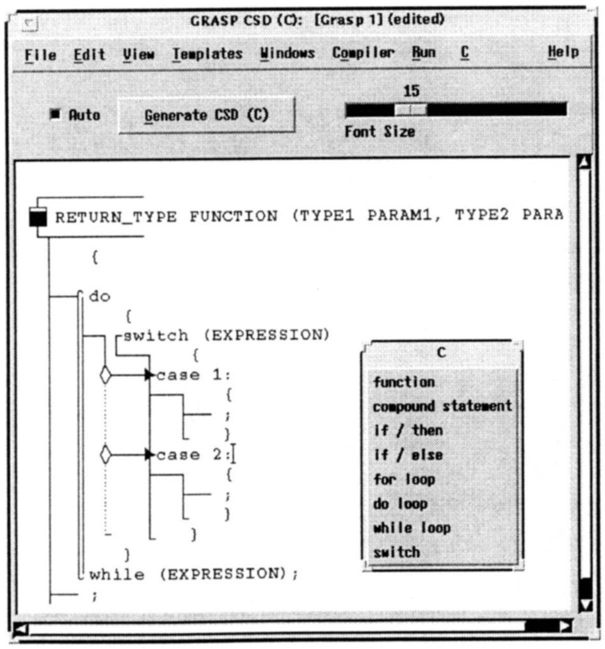
development or maintenance of multi-lingual software systems. GRASP not only provides a common editing and visualization environment for its supported languages, but also provides compilation, make, and execution capabilities. GRASP is coupled with the GNAT Ada 95 and GNU C compilers. The CSD window allows the user to invoke gcc directly for the current program unit to perform a **Make** or a **Compile** and set compiler flags via the Flag Setup dialog box. When an error is reported by the compiler, the offending line of code is highlighted in the CSD window and a message window is raised to inform the user of the nature of the error.

After making an executable, the user may run the file directly from the CSD window by selecting **Run, Run Previous,** or **Run File**. **Run** assumes the user wants to run the executable associated with the source file in the current CSD window. **Run Previous** runs the file that was executed by the most recent **Run** option. **Run File** opens a file select dialog box and allows the user to run any existing executable. The Run Shell Window is opened for input/output to the executing program. This shell runs as a separate process to the execution of the user's program cannot affect GRASP. A separate dialog box that is raised during a Run option allows the user to send various signals to the executing program (e.g., interrupt or kill).

6. The GRASP Model

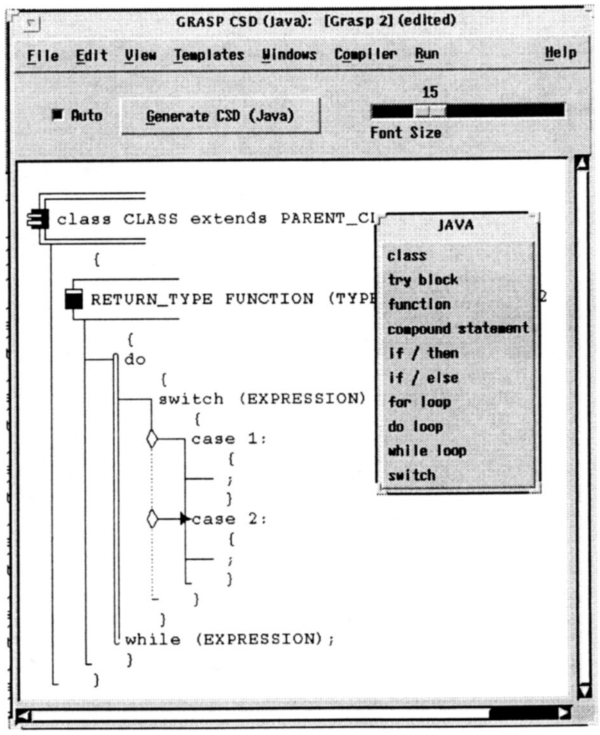
The major system components of the latest full release of GRASP are shown in the block diagram in Figure 13. Currently, the entire prototype is written in the programming language C and runs under Solaris.

The user interface was built using Motif and the X Window System. The CSDgen component controls the generation of control structure diagrams, the CPGgen component controls the generation of complexity profile graphs, and the ODgen component controls the



```
RETURN_TYPE FUNCTION (TYPE1 PARAM1, TYPE2 PARA
{
  do
    {
    switch (EXPRESSION)
      {
      case 1:
        {
        ;
        }
      case 2:
        {
        ;
        }
      }
    }
  while (EXPRESSION);
  ;
```

Figure 11. CSD window for C.



```
class CLASS extends PARENT_CL
{
  RETURN_TYPE FUNCTION (TYP
    {
    do
      {
      switch (EXPRESSION)
        {
        case 1:
          {
          ;
          }
        case 2:
          {
          ;
          }
        }
      }
    while (EXPRESSION);
    }
}
```

Figure 12. CSD window for Java.

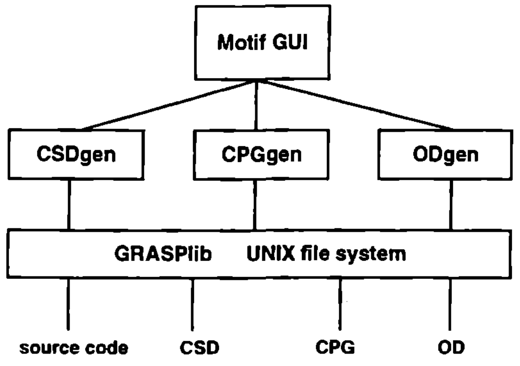


Figure 13. GRASP block diagram.

generation of object diagrams.

The GRASP library component, GRASPlib, supports 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 to facilitate navigation among the diagrams and the production of sets of diagrams.

The control structure diagram generator, **CSDgen**, inputs source code and produces a CSD. CSDgen has its own parser/scanner built using FLEX and BISON, and also includes its own printer utilities. When changes are made to the 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.

The object diagram generation component, **ODgen**, is in the analysis phase and has been implemented as a separate preliminary prototype. The feasibility of automatic layout of object diagrams so that they are both useful and aesthetically pleasing remains under investigation.

6. Conclusion

GRASP is a software engineering tool providing automatic visualization, measurement, and program development environments for Ada 95, C and Java. Other languages such as C++ and VHDL are currently being integrated into the tool. GRASP is being used in five computer science classes at Auburn University and numerous educational and government institutions have downloaded and installed the tool. The most recent prototype has significantly extended the functionality of GRASP. Especially noteworthy are the additions of complexity visualization in the form of the CPG, visualization scaling in the form of Booch module symbols, and support for multiple languages. Preliminary studies (publication currently in progress) indicate that GRASP is highly useful to software engineering efforts and further empirical studies are currently being planned and implemented.

GRASP is freely available via the Internet at the following URL:

http://www.eng.auburn.edu/grasp.html

Acknowledgments

The GRASP project has been supported, in part, by grants from NASA, ARPA, and DISA.

References

[1] Booch, G. and Bryan, D. *Software Engineering With Ada*, 3rd Edition, Benjamin/Cummings, 1994.

[2] Cross, J. H. Improving Comprehensibility of Ada With Control Structure Diagrams. *Proceedings of Software Technology Conference*, April 11-14, 1994, Salt Lake City, UT.

[3] Cross, J. H., Chang, K. H. and Hendrix, T. D. GRASP/Ada95: Visualization With Control Structure Diagrams. *CrossTalk: Defense Software Engineering Journal*, 9, 1, (1996), 20-24.

[4] Cross, J. H., Maghsoodloo, S., and Hendrix, T. D. The Control Structure Diagram: An Overview and Initial Evaluation. (submitted for publication).

[5] Cross, J. H and Hendrix, T. D. Language Independent Software Visualization. P. Eades and K. Zhang (eds.), *Software Visualization*, World Scientific Publishing Company, (1996).

[6] McQuaid, P. A., Chang, K. H. and Cross, J. H. Complexity metric to aid Software testing and maintenance. *Proceedings of Decision Sciences Institute*, 2, (1995), 862-864.

[7] Petre, M. Why Looking Isn't Always Seeing: Readership skills and Graphical Programming," *Communications of the ACM*, 38, 6, 1995, pp. 33-44.