ソフトウェア工学 22－15
（1982. 2. 10）

# An Interactive Support System for SIMPL Program Construction

Morio Nagata, Yoshitake Mishima and Yoshiyuki Shikano
Dept. of Administration Engineering, Keio Univ.
Ken'ichi Harada
Keio Institute of Information Science

An interactive support system have been implemented for the sake of incremental program construction in SIMPL. The SIP (SIMPL Interactive Programming) system obtains information of the user's program from the SIMPL compiler. It provides interprocedural and/or intermodule information during the program creation and modification process at any time. Our approach can be applied to procedure-oriented languages and their compilers.

## 1 Introduction

As one of programming methodologies, it has been proposed that the program should be divided into logical and functional modules. Moreover, incremental programming has come to be recognized as a promising methodology. Here, term of "incremental programming" means a program development process with taking advantage of separate compilation facility in a time-sharing system. Combining these notions, this paper presents an intermodule analysis tool for definition and use of identifiers (variables, procedures and functions), which works cooperating with a compiler.

Really to construct a procedure by using the structured programming language in the incremental programming environment[3], interprocedural information at that state of programming is useful. For example, global variables used in the other procedures are important to construct a new procedure. But it is difficult to obtain such information, and as the state of programming progresses, the configuration of a program always changes. Thus, our attention is concentrated upon interprocedural and/or intermodule information at any stage of the program construction process. Notice that the word 'procedure' is used to mean both a procedure and a function in this paper. We consider that a module consists of one or more procedures and global variable declaration as a compilation unit.

Our system, called the SIP (SIMPL[5] Interactive Programming) system, provides the following interprocedural information in accordance with the programmer's request: attributes and cross reference lists of all identifiers in each module or over modules, data binding list on global variables, call graph representing invocation relationships between each module to the other modules, statistical information of programs, and check list of interface of a module and the other modules. Whenever a programmer wants to get such information, he will use the SIP system interactively.

The main design policy is to provide useful facilities by a little modification to the existing compiler. The SIP system has been implemented by adding only the Line Reference Table (LRT) to the SIMPL compiler.

## 2 An Overview of the SIP System

### 2.1 Structure of the SIP System

The SIP system consists of the SIMPL compiler and the intermodule analyzer. Although the SIP system does not allow any dynamic editting or compiling, an user can get a great many static information from the SIP system. At each time of a compilation, the SIP system collects the internal information of a program given by the compiler. As a result, the SIP system can respond to an user's query on his programs interactively. Figure 1 shows a structure of the SIP system.

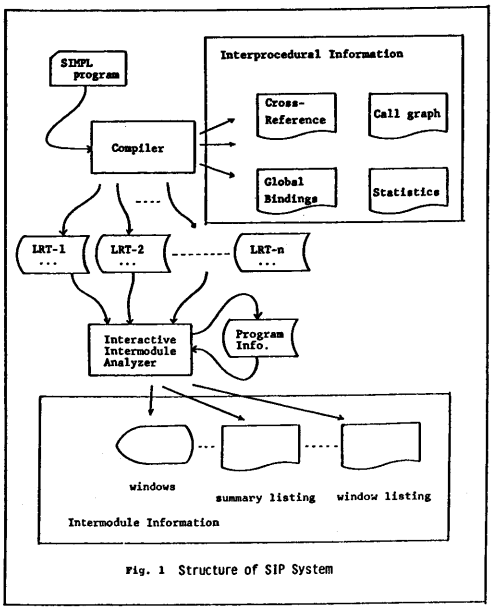


Fig. 1 Structure of SIP System

```
                    Global Data Bindings List

PROC/FUNC           Globals/Bindings -- Line Numbers

PROCFUNCATTR
                    SEGNO
                              [Accessed]  69, 118
                    PLINE
                              [Modified]  71, 72, 75, 77, 78, 83, 91
                              [Accessed]  115
                    BLANK
                              [Accessed]  71
                    CALLTBL
                              [Accessed]  73
                    SYMTAB
                              [Accessed]  74, 78, 80, 99, 104
                    PROCTBL
                              [Accessed]  77
                    PROCFLAG
                              [Accessed]  81
                    ENTRYFLAG
                              [Accessed]  94
                    FWDFLAG
                              [Accessed]  109

GENCALLEDCHAIN
                    SEGNO
                              [Accessed]  130
                    CALLTBL
                              [Accessed]  132, 158
                    REFS
                              [Accessed]  136
                    SYMTAB
                              [Modified]  139, 159
                              [Accessed]  137, 139
                    PROCFUNC
                              [Accessed]  137
                    PROCTBL
                              [Modified]  149, 157
                              [Accessed]  142
                    CALLED
                              [Modified]  147, 151
                              [Accessed]  143, 145
                    GAVAL
                              [Modified]  152
                              [Accessed]  147, 149, 151, 152

PRTCALLGRAPH
                    SYMTAB
                              [Modified]  202, 207
                              [Accessed]  171, 173, 188
                    PLINE
                              [Modified]  172, 174, 180, 182, 190
                              [Accessed]  174, 180, 182, 185, 190, 192
                    COLN
                              [Accessed]  184, 187
                    EXTFLAG
                              [Accessed]  188
                    BLANK
                              [Accessed]  192
```

## 2.2 An Output from the Compiler

On the contrary to a normal compilation, the compiler gives some useful information independently on the other separately compiled modules. The output information with some typical examples is as follows:

1. Cross reference list: The attributes and cross reference list of all identifiers in one program module is generated.

2. Global data binding[1] list: The data binding information on global variables is produced by interprocedural analysis for one program module. As the invocation relationships between procedures have been made clear in a form of internal representations, a binding analysis can determine a point, where a global variable is defined, and where the global is referenced to. Through these analyses, some logical error such as a side-effect with no care can be easily detected.

3. Call graph: By analyzing the invocation relationships between procedures, procedure invocations are represented in a call graph. In general, the structured programs consist of many logical components expressed by procedure. It is very powerful to debug and maintain a relatively large size program that the procedure invocations are exactly examined in their relationships. In addition, after all call graphs of program modules compiled separately are collected, the call graph for the executable (absolute) program would be acceptable by merging all graphs into one. Nothing is more accurate in the invocation order than this complete merged call graph.

```
                Listing of Procedures and Functions

No.   Name                 Calls  Called    Attribute

 1    PROCFUNCATTR             1       1    PROC
 2    GENCALLEDCHAIN           0       1    PROC
 3    PRTCALLGRAPH             2       3    PROC, REC
 4    CALLINGLEVEL             1       2    PROC, REC
 5    CALLGRAPH                3       1    PROC
 6    DUMP                     0       0    PROC
 7    CALLGRAPHGEN             3       0    PROC, ENTRY
 8    PUTGLINE                 0       2    PROC
 9    MANYGLOBALS              0       2    PROC
10    PRTGLOBALBIND            4       1    PROC
11    GLOBALSOFSEG             2       1    PROC
12    GLOBALDATA               2       1    PROC, FWD
13    GLOBALBIND               1       0    PROC, ENTRY


                    C A L L    G R A P H

    0       1       2       3

    CALLGRAPHGEN(316)  **Entry**
            GENCALLEDCHAIN(123)
            PROCFUNCATTR(56)
                    NAME(20)  **Ext**
            CALLGRAPH(241)
                    CALLINGLEVEL(213)
                            CALLINGLEVEL(213)
                    PRTCALLGRAPH(166)
                            NAME(20)  **Ext**
                            PRTCALLGRAPH(166)
                    PRTCALLGRAPH(166)

    GLOBALBIND(534)  **Entry**
            GLOBALSOFSEG(474)
                    GLOBALDATA(51)
                            MANYGLOBALS(431)
                            MANYGLOBALS(431)
                    PRTGLOBALBIND(442)
                            NAME(20)  **Ext**
                            NAME(20)  **Ext**
                            PUTGLINE(382)
                            PUTGLINE(382)
```

4. Program statistics analysis: The general program statistics information such as a number of each statement type, a number of significant tokens, or maximum and average of nesting level are printed in a chart.

## 2.3 Intermodule Information

If the following commands are given by the user, the SIP system provides intermodule information by using the above output from the compiler. Some commands have several subcommands.

1. APPEND: Add current program information created by the SIMPL compiler to the SIP system. If the SIP system has information for the same module, it is replaced by a new one.

2. GET: Specify the module name to which the following commands are to be applied.

3. LIST: Output statistical information of the module specified by the GET command. Any combination of the following information is available.
   --global variables and/or procedures defined
   --entry variables and/or procedures defined
   --external variables and/or procedures referenced
   --number of source lines, statements, proc/func and etc.

   Above statistics can also be obtained for every modules collected in the SIP system instead of the specified one.

4. SEARCH: Search for definition and/or reference points of specified identifiers of the module. This function can be extended over all modules.

5. CHECK: Check the interface consistency of the module against all other program modules. For each external and entry objects, following items are checked by this command.
   -- type of variables
   -- procedure or function type (with type of return value for the function)
   -- number and type of parameters for procedure/function

   This command provides the error detection capability for module interface which can not be gained by the conventional linkage editor and prevents disastrous results at run time caused from erroneous declaration.

6. AFFECT: Point out all module names come under the influence of a modification to specific variables and/or procedures in other module. This function takes advantage of the invocation analysis and is especially useful for checking the affect of modification to be made in the maintenance process.

In addition, an analysis on the global data bindings for all modules can be accomplished by applying a data flow analysis based on the merged call graph.

## 3 Design and Implementation

### 3.1 A View of Our Design

In the bootstrapping process[2] for the compiler enhancement, several useful facilities to support programming activities were thought to be added to compiler itself, if they can be developed with a little modifications to the compiler. The straightforward approach to this problem is to obtain information on the definition and reference of variables reflected with their line numbers in the source program.

Characterizing the overall design of this system, there are a minimal number of the modifications in order to create a skeletal information. The design policies in the modifications are as follows:

1. It is programmed so that the necessary information should be collected only when the compile option is specified.

2. Every information on data items should be packed in a relatively small table, and should reflect all accesses to variables with the line numbers where they appeared.

3. Such a table including the symbol table should be easily separated independently from the compiler.

4. All SIP functions work with the program information library file which contains the set of tables created by the compiler mentioned above. The SIMPL compiler and the analyzer should be loosely connected only by a temporary file containing such tables.

### 3.2 Line Reference Table

We define a line reference table (LRT) that contains the source line numbers with all definitions and references of variables. This LRT is designed to give a skeletal data for the debugging and analyzing facilities. The data items registered in LRT are identifiers (e.g. variable or procedure and function name) and the intrinsic procedures used in a program. Every item includes the line number in a source program where it appears and the indicator whether its value is defined or referenced to. In addition, every procedure binds its scope in LRT.

### 3.3 Implementation

(1) Interprocedural analysis

The compiler at the first stage have already had a variety of testing, debugging and program analysis facilities. These are 1) attribute and cross-reference listing, 2) traces available for line numbers, calls and returns, and variable values, 3) subscript and case range checking, 4) statistics at compile time. In generating a cross-reference listing, especially, the line references have been managed in some LRT-like table. Therefore, we reconstructed this table as the LRT in order to facilitate a generation of cross-reference listing and call graph, and a computation of global bindings.

Call graph: The scanning for procedure names are done interprocedurally only in the LRT, and the invocation relationships are represented in a directed-graph. A node involves both procedure name and referenced line number. While the call graph listing can be provided for every compiled module, this graph is also a subgraph of whole call graph for all separately compiled modules.

Global bindings: The items in the LRT are also scanned interprocedurally with respect to global variables. Definitions and references of variables are listed with line numbers. The globals include both internal and external variables. As the local ones, a reference preceding to an assignment of value can be checked if the flow of control is accurately analyzed through the call graph.

Module interface: As for a program consisting of several modules, the internal LRT and call graph for one module are written on

the library file together with the symbol table. This interface facility automatically generates a call graph, and rearranges the LRT into another LRT which disables an access to local variables and represents the global or external ones in a linear list with lexicographical order.

(2) Intermodule analysis

The SIP system consists of many separate functions[4]. Some of them correspond to the top level SIP commands. After a certain function is invoked, control goes into the subcommand mode of the function and execution continues.

For the purpose of the inter communication of these functions, the system work area (SWA) is prepared in SIP. The SWA contains LRT, intra module call graph and related information only for one module at a time. Intramodule analysis is done using SWA. Some analysis which requires intermodule information should get information of each module from the library file into SWA and create internal table over modules successively.

Table 1 shows the current program size of the SIP system created by the SIP system itself. Currently, as the program size of each SIP function is relatively small and DEC-20 has 256kw (36bits/w) virtual address space, SIP function programs are linked together in the same address space. In the case when the number of functions are increased or each function becomes more complex and larger, segmentation or creation of another address space for each function is required.

## 4 Discussion

The SIP system has been in use by our colleagues and ourselves only a few weeks. However, we conclude that this kind of system is useful for the SIMPL or other procedure-oriented program construction.

During the development of the SIP system, we have been able to detect some errors by using facilities providing interprocedural information. The traditional compiler or linkage editor can not support incremental programming. On the other hand, the programmer can use the SIP system interactively during the incremental programming process.

This system is written in the SIMPL language, therefore the user can modify the SIP system with a little efforts. Consequently, this system is flexible in its user's modification.

In contrast to the separate programming support system, the SIP system effectively uses information obtained by the SIMPL compiler.

Besides the internal form and size of the library file, trade-off between response time for query and memory space plays an important role. Under the time sharing environment, response time have an influence on the system load. Process time for each command should be minimized. For instance, definition and reference list for identifier of each module can not be affected when other modules are modified. On the other hand, call graph over modules should be reorganized each time when invocation of external procedure in certain module is changed.

Table 1 Program Module Statistics for -- PSTAT -- 01/26/82-23:27:35

| | Module-Name | Ext | Cycl. | Lines | Stmt | PROC/FUNC | Ent Var | Ent P/F | Ext Var | Ext P/F |
|---|---|---|---|---|---|---|---|---|---|---|
| 1. | PSTAT1 | SPL | 22 | 325 | 113 | 4/0 | 0 | 2 | 21 | 5 |
| 2. | PSTAT2 | SPL | 49 | 993 | 403 | 12/4 | 0 | 7 | 24 | 7 |
| 3. | PSTAT3 | SPL | 4 | 203 | 32 | 1/0 | 3 | 1 | 2 | 4 |
| 4. | PSTAT4 | SPL | 11 | 251 | 90 | 2/0 | 0 | 1 | 12 | 10 |
| 5. | PSTAT5 | SPL | 7 | 396 | 138 | 5/2 | 0 | 1 | 7 | 8 |
| 6. | PSTATM | SPL | 17 | 226 | 54 | 5/0 | 30 | 3 | 3 | 12 |
| 7. | PSTATU | SPL | 7 | 133 | 36 | 6/0 | 0 | 6 | 3 | 3 |
| | | | Total: | 2527 | 866 | 35/6 | 33 | 21 | 72 | 49 |

In later case, when speed is important, all analyses should be done at each time of modification of modules prior to the next query. In contrast, as it takes time and space, creating call graph at each query is reasonable when such kind of query is in rare.

Only mutable information used by current commands is intermodule call graph. Because of the low frequency of query using a call graph, we decided that it is recreated on each time of recieving the query. In future, it is preferable that user can specify the trade-off between time and space for each command. In such case, system generates internal information required for the commands prior to the query which is classified as time critical by user.

## 5 Conclusion

We have constructed the support system for writing SIMPL programs which consist of a number of modules. However, our approach can be applied to other procedure-oriented languages and their processors with separate compilation facilities.

Finally, the SIP system supports an aspect of human programming activities, and there are tools which also support other aspects. Thus, for the construction of comfortable programming environment, the compiler, the SIP system, a structure editor, an automatic verifier and a dynamic program analyzer should be effectively combined.

## References


1. Basili, V.R. and Turner, A.J., "Interactive Enhancement: A Practical Technique for Software Development," IEEE Trans. on Software Engineering, Vol. SE-1, No. 4(Dec. 1975), 390-396.

2. Basili, V.R. and Perry, Jr. J.G., "Transporting Up: A Case Study," The Journal of Systems and Software 1, Elsevier North Holland, Inc., 1980, 123-129.

3. Medina-Mora, R. and Feiler, P.H., "An Incremental Programming Environment," IEEE Trans. on Software Engineering, Vol. SE-7, No. 5(Sept. 1981), 472-481.

4. Osterweil, L.J., "Draft TOOLPACK Architectural Design," Dept. of Computer Science, Univ. of Colorado at Boulder, 1981.

5. Shikano Y., et al., "System Implementation Language: SIMPL on DEC System-20," Proc. of DECUS Japan, Vol. 1, No. 1(Oct. 1980), 1-27.