©2010 International Journal of Computer Applications (0975 – 8887)
Volume 1 – No. 13

An Ameliorated Methodology for the design of Object Structures from legacy ‘C’ Program

Dr. Shivanand M. Handigund
Dept. of Computer Science & Engineering.
Bangalore Institute of Technology
Bangalore –560 004

Rajkumar N. Kulkarni
Dept. of Information Science & Engineering
Ballari Institute of Technology & Management
Bellary – 583 104

ABSTRACT

Information systems of many organizations are processed through system of interrelated ‘C’ programs. Since, the ‘C’ programming language was developed in the early second half of the last century. It couldn’t incorporate to facilitate the current day’s technology. Therefore, the programs developed based on this are not coping with the advancement of technology. There is a need to harness the useful business information buried across the legacy ‘C' systems and the advancement in the information technology. This act of harnessing the old virtues in new environment is like resolving the labyrinth. The paper proposes a way of resolving this deadlock situation by reverse engineering the legacy ‘C’ systems into the design specifications of the target environment, and then forward engineering the target design specification into the desired language code.

This paper attempts to develop a reengineering methodology that automatically abstracts the view elements like attributes, functional dependencies, interrelationships between group of attributes and actor’s interface, etc. The correctness and completeness of these abstractions are ensured using Unified Modeling language (UML) diagrams. The methodology blends the reverse engineering and re-design stages into a unified process.

Keywords:

Functional dependencies, abstraction, reengineering, business rules, legacy systems, reverse engineering, system requirement specification.

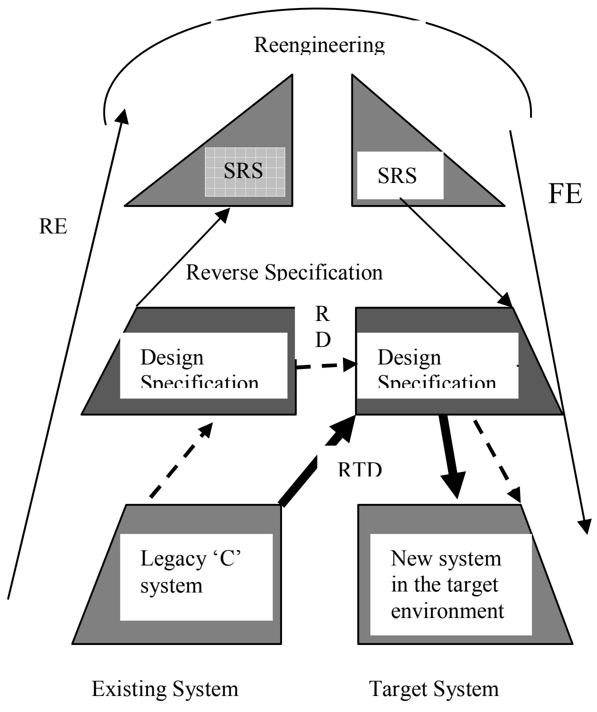
1. INTRODUCTION

Information systems of many organizations are processed through system of interrelated ‘C’ programs. Since, the ‘C’ programming language was developed in the early second half of the last century. It couldn’t incorporate to facilitate the current day’s technology. Therefore, the programs developed based on this are not coping with the advancement of technologies in the areas of Storage, Processing, Graphical user Interfaces. There is a need to harness the useful business information buried across the legacy ‘C' systems and the advancement in the information technology. This act of harnessing the old virtues in new environment is like resolving the labyrinth. The following are the different ways to resolve labyrinth.

- One way is to discard completely the old system and develop a totally new system. This approach suffers from a pitfall of loosing useful business rules accumulated throughout the development and maintenance process. Thus, it will not serve the purpose of developing the new system by incorporating the buried business rules.
- The second approach is the translation of the existing ‘C’ code into the target language. This translation process can be carried out in two different ways viz. manual and automatic approaches. The manual approach is time consuming and error prone approach. Moreover, the enormity of the stored legacy system compels the transformation process next to impossible. The automatic translation can be performed by developing a translator tool, because of the flexibility involved in the program coding the translator can’t be developed with correctness and completeness. Thus translator may not translate the entire code. It may translate the simple code leaving it to human to translate complex code. Thus the translation of the legacy system directly to the target language may not be a good solution for the reuse of legacy ‘C’ systems. Moreover the translation always takes the taste of the original language.
- The third approach is the wrapping. In this approach, the old system is wrapped through the use of emulators so that at the front end the new system is running but at the back end the same old system is running with the slow phase. Thus, the wrapping approach is also not so useful for the reutilization of the existing code.

The best alternate approach is the reengineering of the old system to the new target system. We propose a new methodology in which we abstract the view elements of the new system from legacy ‘C’ system by blending the reverse engineering process with re-design process to form a unified process which is shown in **figure 1**.

The authors D.B. Pathak and Shivanand Handigund [1] developed a similar methodology for legacy COBOL systems. In their methodology they have abstracted the view elements of the design specification from the old system and then migrated to the new system. This proposed methodology is an ameliorated one over their methodology and also which changed legacy system.



RE→ Reverse Engineering
FE → Forward Engineering
RD →Re-Design
RTD→ Reverse Target Design

- - - ▶ In the Existing Work

━━▶ In the Proposed System

Figure – 1. Specific Model for reengineering

2. TERMINOLOGY

Referenced Attribute – A variable is said to be referenced in a statement if the value of that variable is used during the execution of the statement. For ex., A = B + C. The values of B & C are used or referenced in the statement [1, 16]. The attribute value is unaltered with the execution of the statement.

Defined Attribute – A variable is said to be defined, in a statement if the execution of that statement can alter the value of the variable. For ex., A = B + C. The values of B & C are referenced or used in the statement, and A is said to be defined [1, 16].

Preserved Attribute – A variable whose value is unaltered with the execution of statement, then the attribute value is preserved. [1]. A variable may be both referenced and defined in a statement, or may be both preserved referenced, but cannot be preserved and defined in a statement.

Functional Dependency - If R is a relation schema, and A and B are non-empty sets of attributes in R, then B is functionally dependent on A, iff each value of A in R has associated with it exactly one value of B in R, and the formal notation would be A → B, where A is referred to as the determinant, and the attributes on RHS are referred to as the dependent. A → B is formally read as "A functionally determines B" [10].

Closure of Functional dependency set: The closure of functional dependency set is formed with set of all functional dependencies present and the set of all derived implicit functional dependency set.

Legacy Software: The Legacy software is defined as a 10 to 15 year old software system, which is still executable but resists for modifications leading to software crash.

The characteristics of the legacy software are [1]:
- The legacy software is large, typically with millions of lines of code
- It is more than decade old
- It is written in legacy languages like C, COBOL, FORTRAN etc.
- It is unstructured and badly documented
- It is often **mission-critical** at work in the business organization

Mission Critical: The Information System cannot run without the presence of software even for a short period.

Abstraction Levels: The abstraction level is defined with respect to the proximity to the machine understanding. The abstraction levels (Requirement, Design, Implementation) corresponds to a phase in the software development life cycle and defines the software system at a particular level of detail [1].

Software maintenance: The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment [1].

Restructuring: Restructuring is the transformation from one representation form to another at the same abstraction level. The transformation preserves the external behavior of the system. Restructuring is typically used in implementation stage to transform code from an unstructured form to a structured form [1].

Reverse Engineering: The reverse engineering is the process of analyzing the subject system with two goals [1]:
- To identify the system's components and their interrelationships
- To create representations of the system in another form at a higher abstraction level.

Forward engineering: Forwarding engineering is the traditional process of moving from the requirements of the system to its design stage, and from design stage to the concrete implementation of the system. It should be preceded by the reverse engineering in which case it is simply called as software development [1].

Reengineering: The process of re-engineering can be defined by the simple formula [1]:

Re-engineering = Reverse engineering + Change in techniques + Forward engineering

3. PROPOSED METHODOLOGY

'C' programs do not have strict indentation rules. Because of the flexibility available in the 'C' programming language multiple statements can be placed on a single line or a single statement can span several lines [18]. All 'C' statements must be terminated with a semicolon and the last statement in the body of the loop may terminate with right brace. Similarly the beginning of the statement follows either semicolon and a blank or left brace. So, in the beginning of the process, we are assigning the line numbers to each statement of the 'C' program except for the blank lines and the comment lines [12, 13, 14, 15]. This serve as a moulded input for the next steps in the abstraction of functional dependencies.

The methodology for the abstraction of functional dependencies and the design of object structures from the 'C' program is explained in the following steps:

3.1 ABSTRACTION OF CONTROL FLOW GRAPH FROM 'C' PROGRAM

The Control Flow Graph (CFG) indicates the execution control flow of the entire program or system of programs. In 'C' program, each line consists of one statement or control predicate. To represent it as a graph we have to assign one vertex (node) for each statement. The control flow between the statements is indicated by an edge between those vertices. Since, the Legacy 'C' system contains thousands of lines of code, and it is a lengthy process to represent the CFG. The memory requirement to represent CFG is also more when the size of the program increases. This can be reduced by collapsing the consecutive straight line sequential statements into a single block and representing it as a vertex. The CFG is stored in buffer in the form of Control Flow table, in which the first and second column contains statement numbers of start and end statements of collapsible straight-line sequence of basic block. If the block contains the single statement, the first & second columns contain the same statement number. The third and fourth columns contain statement numbers of alternate control transits. Here, IF - ELSE, FOR, SWITCH, WHILE, DO-WHILE, EXIT, RETURN, CONTINUE, and FUNCTION CALLS are treated as verbs to enable the parser to identify the beginning of each statement separately. The end of the program is represented with the statement number followed by the character **E**.

The following paragraph depicts how the CFG works for "**if**" and "**for**" statement.

if Statement: The "**if**" statement may consist of a single statement, a block of statements, or nothing (in the case of empty statements). The **else** clause is optional. **if expression** evaluates to true, the statement or block that forms the target of **if** is executed; otherwise, the statement or block that is the target of else will be executed, if it exists. The scope of **if** terminates either with a single statement following **if** conditional, which is ending with semicolon or a set of statements enclosed in a pair of braces. Similarly the scope of **else** terminates with a single statement following **else** which is ending with a semicolon or a set of statements enclosed in a pair of braces. The statement numbers of these alternate transits are entered in the **column 3 and column 4.** We are using column 3 for the TRUE value of the alternate transit and column 4 for the FALSE value of the alternate transits.

for statement: The **for** loop is another entry-controlled loop that provides a more concise loop control structure. The execution of the **for** statement is as follows:

- Initialization of the control variables is done first, using assignment statements.
- The value of the control variable is tested using the test-condition. The test-condition is a relational expression that determines when the loop will exit. If the condition is **true**, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop. The statement numbers of these alternate transits are entered in the **column 3 and column 4**. We are using **column 3** for the TRUE value of the alternate transit (when the test-condition is true) and column 4 for the FALSE (when the test-condition is false) value of the alternate transit.

We have already developed the automated methodology in the form of a tool [14, 15]. The CFG of the sample C program depicted in **Figure 2** is shown in **Table 1** as follows

Table 1. The Control Flow Table for the Program in Figure 2

START	END	Transition 1 / Next Jump	Transition 2 / Alternate Jump
1	14	15	19
15	18	14	
19	23	24	33
24	32	23	
33	34**E**		

3.2 ABSTRACTION OF DATA FLOW GRAPH FROM 'C' PROGRAM

The above designed CFG is used to design Data Flow Graph (DFG) in the form of Data Flow Table (DFT). Each entry in the DFT indicates the referenced and defined items of that statement. These entries are organized in the CFG order. The sample entries are shown in the **Table 2**. The data items in a 'C' program are defined by scanf, fscanf, gets, fread statements, the left side attribute of the arithmetic expressions except stdin, and stdout. The data items in a 'C' program are referenced by printf, fprintf, puts, fwrite, statements, the right side attributes of the arithmetic expressions and the attributes of control predicates. The DFG for our sample 'C' program depicted in figure 2 is represented in **Table 2**.

Table 2. The Data Flow Graph for the Program in Figure 2.

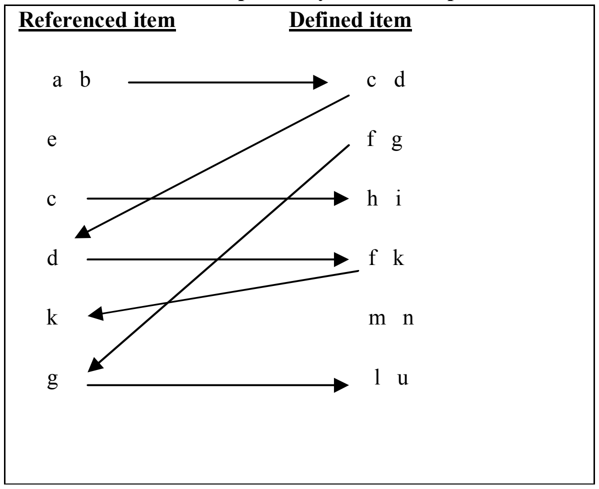
Statement Number	Referenced variable	Defined variable
1	---	---
--	---	---
10	---	---
11		filename
12	filename	fp
13	---	---
14	i	i
15	---	---
16		ENAME,BS,LIC, PT,IT,OA
17	ENAME, BS, LIC, PT, IT, OA	
18	---	---
19	fp	
20	---	---
21	filename	fp
22	--	---
23	i	i
24	---	---
25		ENAME, BS,LIC,PT,IT,OA
26	BS	HRA
27	BS	DA
28	BS,DA,HRA,OA	GS
29	LIC,PT,IT	DED
30	GS,DED	NS
31	BS, DA, HRA, LIC, PT, IT, OA, GS, DED, NS	
32	---	---
33	fp	
34	---	---

3.3 ABSTRACTION OF FUNCTIONAL DEPENDENCY SET AND THE FD CLOSURE FROM ‘C’ PROGRAM

If the statement contains both referenced and defined items, then the referenced items determine the defined items. Thus there is a functional dependency between the referenced and defined items.

The Closure is formed by combining attributes of the statements connected by referenced defined cycle as shown in **Table 3 below:**

Table 3. Functional Dependency set of a sample DFT



Here the functional dependency set includes:
a b → c d
e → g
c → h i
d → f k
g → l u

Such Functional Dependency sets are to be identified from the closure of the functional dependencies as shown in **figure 3**. Thus from the functional dependency set, we obtain the attributes participating in the functional dependency set. This cycle repeats for all the variables in the data flow graph table.

The proposed tool developed here abstracts the functional dependencies from the input ‘C’ program shown in **figure 2.**

The functional dependencies abstracted from the program are:

BS → HRA
BS → DA
BS, DA, HRA, OA → GS
LIC, PT, IT → DED
GS, DED → NS

The Functional Dependency Set closure for the above functional dependencies is:
{BS}+ → {HRA}, {BS}+ → {DA}
{BS, DA, HRA, OA}+ → {GS}
{LIC, PT, IT}+ → {DED}
{GS, DED}+ → {NS}

3.4 ALGORITHM FOR THE ABSTRACTION OF FUNCTIONAL DEPENDENCIES FROM ‘C’ PROGRAM

/* Algorithm for the Abstraction of Functional Dependency from the ‘C’ program */
Input: Executable ‘C’ program
Output: Functional Dependencies

1. **[Moulding of input ‘C’ program]**
   1.1 Assign Line number to each physical statement of the program except comment lines and blank lines.

2. **[Abstract the Control Flow Graph of the Program]**
   2.1 Abstract the Control flow of the 'C' program and store it in the form of a Control Flow Table as shown in table 1.

3. **[Abstract the Data Flow Graph of the Program]**
   3.1 For each statement of the input program, identify the referenced variable and the defined variable.
   3.2 Write separately the statement number, referenced variables, and defined variables.

4. **[Abstraction of Functional Dependencies]**

```
For all variables defined in the Data Flow Graph of step 3
Repeat
    Find the Closure of functional dependencies as
    explained in section 3.3 until, for all the
    variables in the defined and referenced column
    of the Data Flow Graph.
```

4. CASE STUDY

The proposed procedure is implemented for number of 'C' programs and the results we got are correct and complete. The sample 'C' program depicted in **figure 2** is the output of moulding process.

```
#include <stdio.h>
#include <conio.h>
main()
{
FILE *fp;
int number,
quantity,BS,DA,HRA,OA,LIC,PT,IT,GS,DED,NS,i;
float price,value;
char ENAME[10], filename[10];
clrscr();
printf("Input file name\n");
scanf("%s", filename);
fp = fopen(filename, "w");
printf(" ENAME   BS   LIC   PT  IT   OA\n");
for(i=1; i<=3; i++)
{
fscanf(stdin, "%s %d %d %d %d %d", ENAME, &BS,
&LIC, &PT, &IT, &OA);
fprintf(fp, "%s %d %d %d %d %d", ENAME, BS, LIC, PT,
IT, OA);
        }
fclose(fp);
fprintf(stdout, "\n\n");
fp = fopen(filename, "r");
printf("ENAME    BS  DA  HRA  LIC  PT  IT  OA
GROSS  DED  NET_SAL \n");
for(i=1; i<=3; i++)
{
fscanf(fp, "%s %d %d %d %d %d", ENAME, &BS, &LIC,
&PT, &IT, &OA);
      HRA = BS * 0.09;
      DA = BS * 1.14;
      GS = BS + DA + HRA + OA;
      DED = LIC + PT + IT;
      NS = GS - DED;
fprintf(stdout, "%-8s %5d %4d %4d %4d %4d %4d %4d
%5d %5d %5d\n",
                ENAME, BS, DA, HRA, LIC, PT, IT, OA,
GS, DED, NS);
}
fclose(fp);
}
```

Figure 2. A sample 'C' Program

5. CONCLUSION

This paper presents an automatic tool that abstracts attributes and forms the first cut object classes through the abstraction of functional dependencies. This procedure is implemented by a scenario of methods to abstract control flow graph and then to represent the data flow graph in the form of data flow table. Using the data flow table of the program and data & control dependencies concept, the functional dependencies are linked to form the closure. Repeating this procedure, different functional dependency sets are obtained, and then the implied functional dependencies are eliminated using the axioms of functional dependencies. The attributes within each closure group form the first cut object structures. The correctness and completeness is authenticated by the use of axioms.

6. REFERENCES

[1] Shivanand M. Handigund, "Reverse Engineering of Legacy COBOL systems", Ph.D. Thesis, 2001, IIT Bombay, Mumbai.

[2] Ronald S. King, James J. Legendre, "Discovery of Functional and Approximate Functional Dependencies in Relational Databases", Journal of Applied Mathematics And Decision Sciences, 7(1), 49-59, 2003.

[3] Wie Ming LIM, John Harrison, "Discovery of constraints from data for Information system Reverse Engineering", IEEE 1997, 39-48.

[4] Wie Ming LIM, John Harrison, "Parallel approaches for Discovering Functional Dependencies from Data for Information System Design Recovery", IEEE 1997, 254-260.

[5] Victor Matos, Becky Grasser, "SQL-based Discovery of Exact and Approximate Functional Dependencies", SIGCSE Bulletin, Volume 36, Number 4, Dec-2004, 58-63.

[6] Hong Yao, Howard J. Hamilton, and Cory J. Butz, "FD_Mine: Discovering Functional Dependencies in a Database Using Equivalences".

[7] Jalal Atoum, Dojanah Bader, and Arafat Awajan, "Mining Functional Dependency from Relational Databases Using Equivalent Classes and Minimal cover", Journal of Computer Science 4(6): 421-426, 2008, Science Publications.

[8] Iztok Savnik, Peter A. Flach, "Bottom-up Induction of Functional Dependencies from relations", Knowledge Discovery in Databases Workshop WS-93-02, 174-185.

[9] Herbert Schildt, "C The Complete Reference", Fourth Edition, Tata McGraw-Hill Publishing Company Limited, New Delhi, 2000.

[10] Julian M. Scher, Canghui Qiu, "FD-EXPLORER: A pedagogical and Design Tool for Functional Dependency Exploration", in the proceedings of ISECON 2004, v21,

1-7.

[11] Mannila, H., and Raiha K.J., "Algorithms for Inferring Functional Dependencies from relations", Data and Knowledge Engineering, 12(1): 83-99, 1994.

[12] Rajkumar N. Kulkarni and Shivanand M. Handigund "Abstraction Of Structural Components From Legacy 'C' Program", International Conference on "Advances in Computer Vision and Information Technology (ACVIT – 07)", Aurangabad, India, November 2007, pp. 1523-1530.

[13] Rajkumar N. Kulkarni and Shivanand M. Handigund, "Moulding The Legacy 'C' Programs For Reengineering", International Conference on "Advances in Computer Vision and Information Technology (ACVIT -07)", Aurangabad, India, November, 2007, pp-1531-1537.

[14] Rajkumar N. Kulkarni and Shivanand M. Handigund, "Abstraction of Structural and Behavioral Components from Legacy 'C' Program", 2nd International Conference on Advanced Computing and Communication Technologies (ICACCT 2007), Panipat, Haryana, India, November, 2007, pp 550-554.

[15] Rajkumar N. Kulkarni and Shivanand M. Handigund, "Abstraction Of Structural And Behavioral Components From Legacy 'C' Program", International Journal of Computing Science and Communication Technologies, Vol. 1, No. 1, July 2008, pp 70 – 75.

[16] K K Aggarwal and Yogesh Singh, "Software Engineering", Revised second edition, New Age International(P) Limited, 2005, New Delhi.

[17] E. Balaguruswamy, "Programming in ANSI C", third edition, Tata McGraw-Hill Publishing Company Limited, New Delhi, 2006.

[18] T. D. Brown Jr., "C for Basic Programmers", Tata McGraw Hill Publishing Company Limited, New Delhi, 1992.