Defence Science Journal, Vol 43, No 2, April 1993, pp 121-127

A CASE Tool to Create an Object-oriented System

S. Srikumar and V. Rajaraman

Computer Science and Automation, Indian Institute of Science, Bangalore-560 012

ABSTRACT

This work is concerned primarily with implementing a CASE tool to convert an application specified by using data flow diagrams to an object-oriented program. An integrated set of tools and techniques for design, development and implementation of software for business applications is presented. The methodology uses primarily data flow diagrams alongwith decision tables. The object-oriented programming techniques have been applied in the methodology for developing the applications using these tools. An inheritance diagram has been evolved. A data flow diagram processor has been developed which generates an equivalent C++ program. A graphical user interface to paint the data flow diagram has been developed. A decision table processor that generates a C++ program has also been developed.

1. INTRODUCTION

There has been a growing need for data processing, as many organizations are computerising their operations. With the advent of powerful workstations, there has been a growing interest in computer-aided software engineering (CASE). This paper deals with an application generator on the lines of the CASE methodology using object-oriented approach.

Work on the same lines was done by Gopikrishna and Rajaraman[1]. The specifications were presented in the form of data flow diagrams[2], which are encoded in a language BDPas (Business Data Processing Pascal). Their work generated a Pascal program. In the present work also, the specifications are given as data flow diagrams. A graphical user interface has been developed using which the user may draw the diagrams on the screen instead of encoding them. The user enters the details of each process using the graphical user interface (GUI). Once this is complete, the data flow diagram preprocessor is called and the code is generated in C++. There is also a facility to embed mixed entry decision tables in the code. Another advance in the current work is the use of object-oriented programming[3] in the methodology for developing the system.

2. OVERVIEW OF THE CASE TOOL

Block diagram of the CASE tool designed by the authors is shown in Fig. 1. The tool can be divided into two parts. One part is a decision table processor or the decision table converter. It takes a decision table in a format specified in Fig. 2 as input and produces the C++ function for that table. The C++ function is directed to the standard output and can be redirected to a suitable file, which can be included in the application. The second part of the tool is the data flow diagram

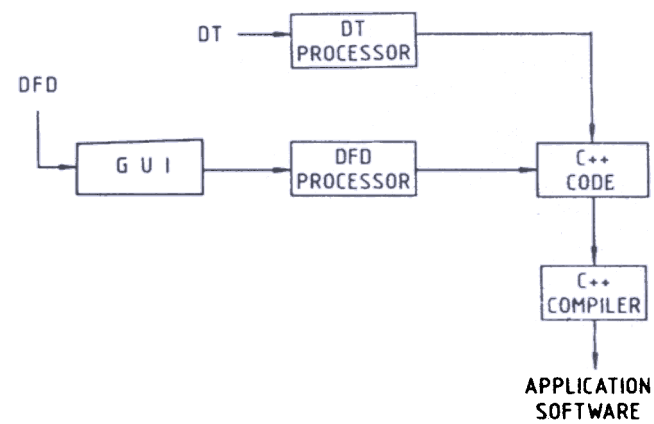

Figure 1 Block diagram of CASE tool.

Received 26 October 1992

processor and the GUI. The data flow diagrams are drawn using the GUI in a top down fashion. The top level data flow diagram is the first diagram.

```
no-of-conds no-of-rules no-of-actions else-present?
name : < name >
parameters :
< parameter > [, < parameter >
conditions
< conditions >
actions
< actions >
```

Figure 2. Format of the decision table for the preprocessor.

3. USE OF OBJECT-ORIENTED APPROACH

One of the advantages of using the data flow diagram is the identification of reusable functions. These functions have to be data-type independent. There should preferably be no globally declared variables. Two types of data structures that are commonly used in business data processing are the array and file. Thus, we can treat the array and file as objects. The functions related to business data processing that can be used for an array and a file are listed in Table 1. It is seen from this table that similar operations are done for an array of records and a file of records. Operations like create, sort, search and delete are common to both. The algorithms used in these operations are the same whether it is an array or a file. The only difference is the object on which these operations are done.

Table 1. Functions used in business data processing

Function name	Description
CRTFILE	Create a file of records
FSORT	Sort a file
FSEARCH	Search for a record occurrence in a file
FBATCH	Batch all record occurrences satisfying a condition in a file
FAPPEND	Append a record after a particular record in a file
FDELETE	Delete a record occurrence in a file
CRTARRAY	Create an array of records
SORT	Sort an array of records
SEARCH	Search an array for a record
APPEND	Append a record after a particular record in an array
BATCH	Batch all record occurrences in an array
DELETE	Delete a particular record in an array

Hence, all these functions can be placed in a common object from which both the array object and the file object can be derived. This common object is called by us the *base object*. Figure 3 shows the inheritance diagram. The file object is split into more objects. All these objects, except the base object and the filevars object, are available to the user. For example, all the files in the user's system can be derived from read-write file.

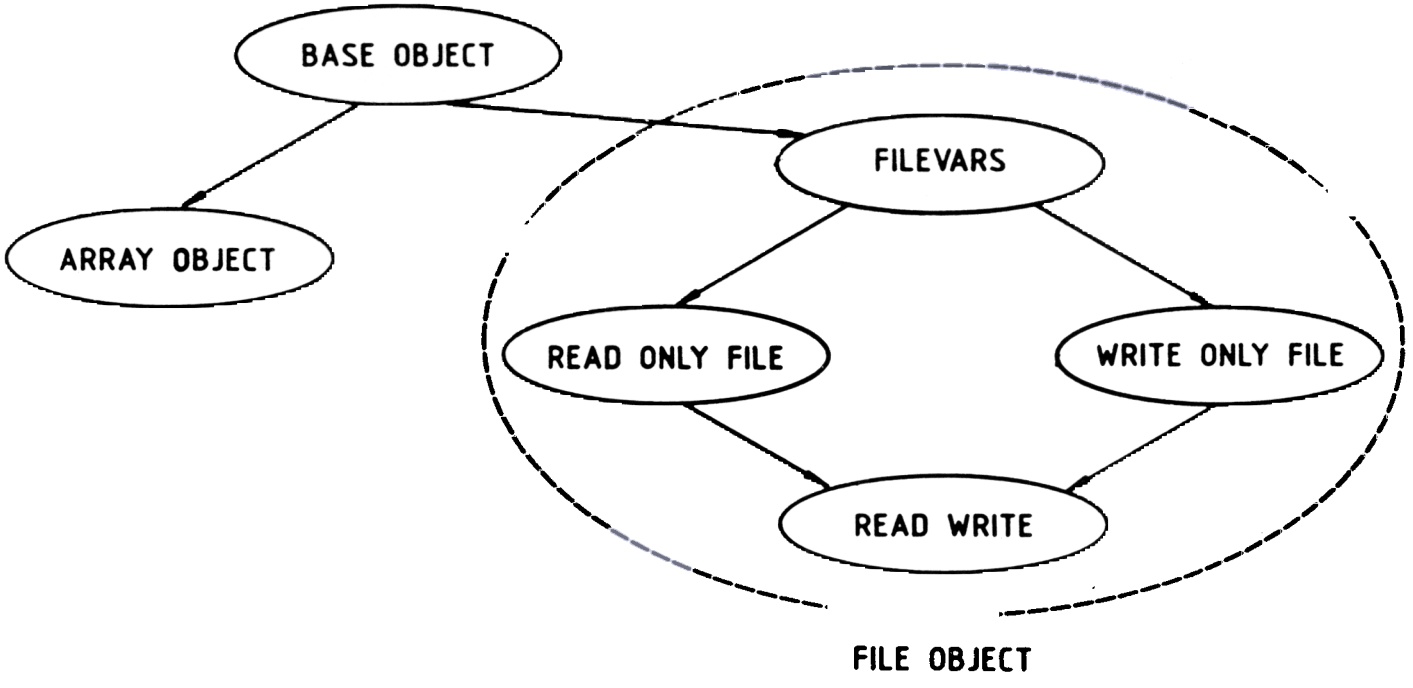

Figure 3. Inheritance diagram.

The functions shown in Table 1 have to be used on the right object. In other words, the correct object will have to be *read from* in the case of, say, the function search and *written into* in the case of, say, the function create. This is taken care of by creating two functions, get-record and store-record, in the base object and making them virtual. These two functions will be defined in the array object and file object. In addition, virtual functions such as position (to position at a particular record) and get-position (to get the current position) have also been defined in the base class.

The objects have been coded in BORLAND C++ in MS-DOS.

4. DECISION TABLE PROCESSOR

Decision table is a tool used to specify a complex decision procedure[4]. A decision table preprocessor or decision table converter (DTC) has been developed which takes a mixed entry decision table and generates a C++ function for that table. The format that the DTC understands is given in Fig. 3. The first line of the decision table contains four integers. These are the number of conditions, number of rules, number of actions and 0 if else is not present, 1 otherwise. Next, the name of the decision table should be present. If it is not present, a default name is assigned. A warning is issued in this case. There could be a clash in the default name if there is more than one decision table. Hence, it is best to give a name to the decision table. The parameters, if any, should come next. These parameters are printed as the parameters of the function. The conditions and actions appear next. The entries in the condition entries and action entries are separated by a colon. The conditions and actions are identified by the keywords conditions and actions respectively. A decision table has to have conditions and actions. If either of these are missing, then it is a fatal error. The DTC stops further processing and quits after issuing an appropriate error message. If the decision table is correctly coded it is converted to a fuction in C++ using the mask method. This fuction can be included and calle from the main application.

The algorithm used in the DTC is as follows:

Step 1: Scan the first line and read the number of conditions, number of rules, number of actions and the else present (boolean).

Step 2: Read the name of the decision table, if present. This name will be the function name of the decision table. If a name is not present, a default name, *dectable0,* is given. However, there will be a clash if there is more than one decision table. A warning is issued when the default name is given.

Step 3: Read the parameters of the decision table. The parameters become the arguments of the function.

Step 4: Process the conditions. If the keyword conditions is not present, then it is a fatal error and the DTC stops further processing and quits. The conditions are processed in the following way.

As mentioned earlier, the mask method[4] is used in the resulting function. The *H* matrix is first generated. The dimension of the *H* matrix is the number of conditions by number of rules.

For each condition, do the following.
For each rule in the condition do the following.

1. If there is a dash for the corresponding entry of the rule, that element of the matrix is assigned 1 (true).
2. If there is a Y or N entry, then that element of the *H* matrix is assigned0. It is ORed with one of the following:
   (a) If it is a y (yes) entry, then the result of condition testing.
   (b) If it is a n (no) entry, then the not of the result of condition testing.

The above combines steps 1 and 2 of the algorithm for the mask method[4].

Step 5: Process the actions. Each rule is generated as a case statement. The correct rule to be applied is obtained by calling the function *find-rule-number.* This function takes in the condition matrix (*H*), the number of conditions and the number of rules and returns the correct rule number or – 1 in case of ambiguity. The algorithm followed for the conversion is as follows.

For each rule,

For each action,

1 If the entry for the action for that rule is a –, do nothing.
2. If the entry is a cross (X), generate the action statement.
3. Otherwise signal an error.

The code is directed to the standard output. It can be redirected to a suitable file which can be included in the application. The **DTC** has been written in ANSI C and the function *find-rule-number* has been written in BORLAND C++.

5. GRAPHICAL USER INTERFACE

The GUI has been developed using X Window Programming under UNIX[5,6]. This is an interface for drawing data flow diagrams. The interface is made up of three windows, a selection window, a drawing window, and a message window (Fig. 4).

The selection window contains the option *create* for keying in the included files, user defined functions, global variable declarations, local variable declarations, and object declarations. A pull down window appears when the *create* option is pressed. On choosing the correct option in the pull down menu, a dialog box appears which asks for a file name. For example, if the user wants to input, say, include files, he creates a file in which he places the include files. The name of this file has to be entered in the dialog box. In other words, the include files, user defined functions, etc. are placed in respective files. On choosing the option, the file name is given. The other options include saving the diagrams, creating the diagram and creating the C++ code. All selections are done by the left mouse button.

The middle window is the drawing window where the user draws the diagrams in a top down fashion. As the user draws the diagrams on the screen, these diagrams are stored in a linked list which contains the following fields: type of object (whether it is a bubble, or a line, etc), the coordinates of the object and if it is a text object, the text and its length. To enter the details of the bubble, the user presses the right mouse button. A dialog box appears in which the user enters the file name where the details of the bubble are present. The middle button redraws the screen.

The message window is only an output window where error or warning messages are displayed.

The design of the graphical user interface is as follows. All the three windows discussed above are children of the root window. Each option in the selection window is a child window of the selection window. The pull down menu is a child of the root window, as it is too big to be made the child of the selection window. However, the window is positioned appropriately, so that it appears as part of the create window of the selection window.

6. DATA FLOW DIAGRAM PROCESSOR

Data flow diagrams are used in the design phase to depict the overall logic of a system. A data flow diagram models a system by using external entities from which data flows to a process which transforms the data and creates output data flows which go to other processes or external entities or files[2]. The main merit of data flow diagrams is that it can provide an overview of what data

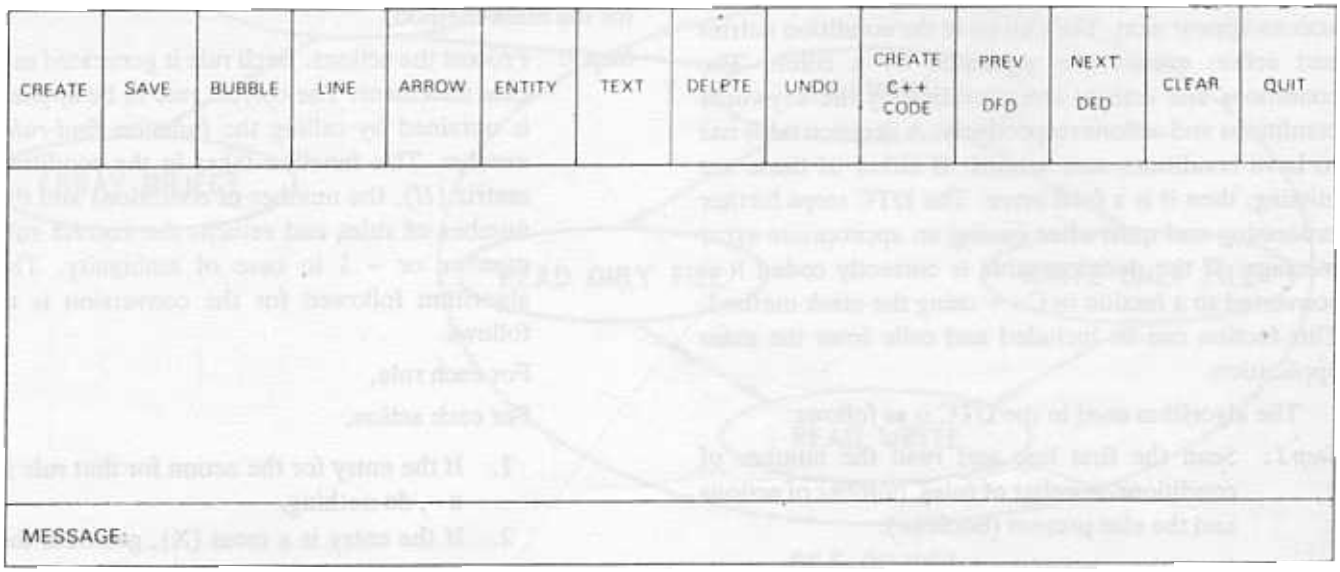

Figure 4. The graphical user interface

a system would process, what transformations of data are done, what files are used and where the results flow.

A graphical user interface has been developed to paint the data flow diagrams. Details of each process are entered. When all the required details are entered, the application is generated which can be compiled using a suitable C++ compiler to generate the executable code.

The data flow diagram processor captures the interconnection diagram on the graphical user interface onto a data structure shown in Fig. 5. These diagrams are drawn in a top down order. The top level data flow diagram is the first diagram. Each data flow diagram is coded as a function, the top level data flow diagram being main(). The data flow diagram first generates the object definitions. These objects are entered using an option in the graphical user interface. Then the application code is generated.

The algorithm used for the generation of the code is given below. The data structure (Fig. 5) is used to generate the code. The application code is generated in a file called *main . cpp.*

Step 1: The user defined functions are generated first. The linked list is iterated from the beginning to the end. These functions are generated in a file called *userfuncs.cpp,* which is automatically included in the final application.

Step 2 The object definitions are generated. This is done in the following way. Two files are generated for each object. Each file name will have the object name and an extension. This object name is part of the structure for the object linked list. The first file is the declaration file ending in *.h* and the other is the definition file ending in *.cpp.* The declaration file contains the variable and method declarations of the object. The definition file contains the definitions of all the methods declared in the declaration file. The declaration file is included in the definition file and the definition file is included in the main application.

Step 3 The include files for the application are generated. The include file linked list is traversed from the beginning till the end and generated. The include files are placed in the beginning of the application file *main.cpp.*

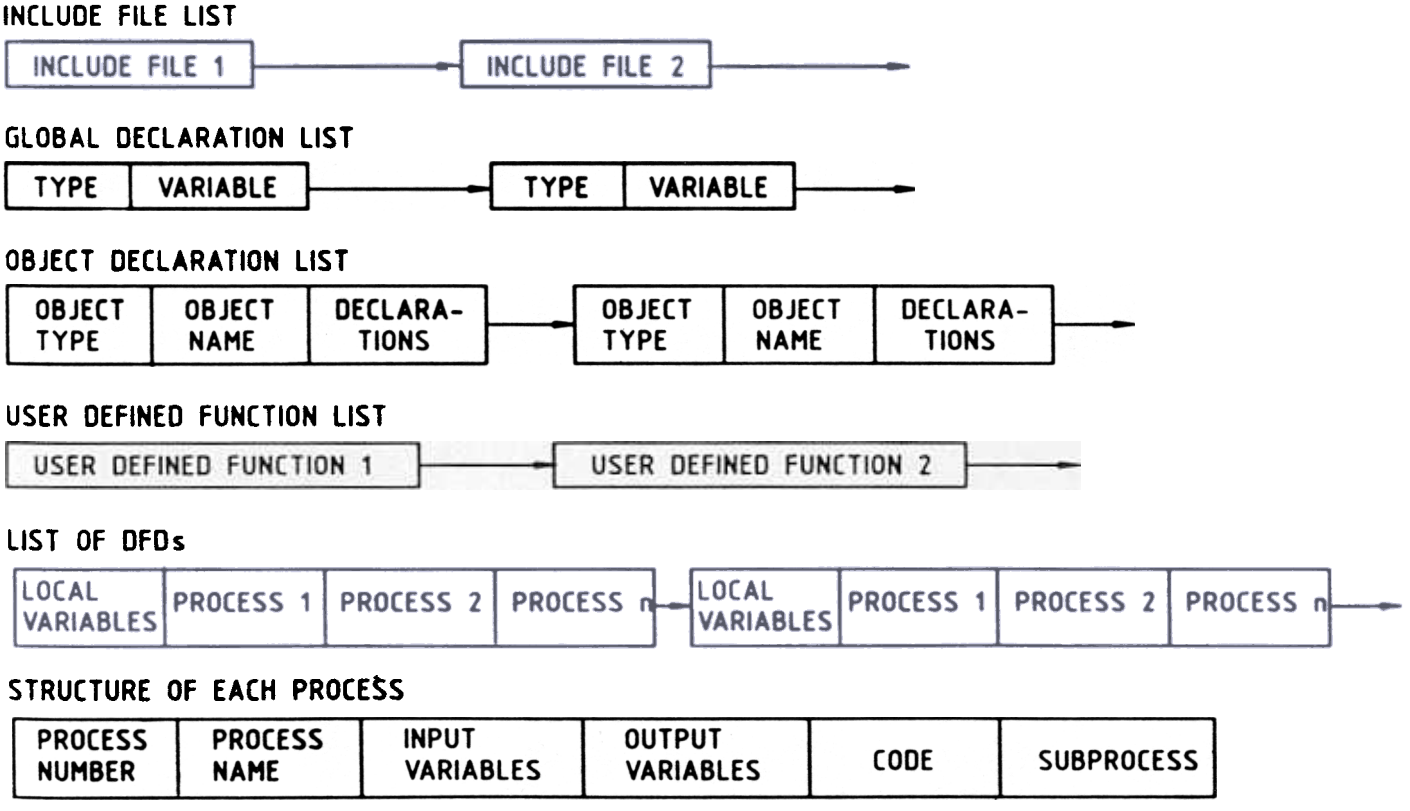

Figure 5. Data structures used

p4: The globals are generated using the global declaration linked list. These globals are placed immediately after the include files.

p5: Finally, the application source code is generated. Each data flow diagram is coded as a function. The top level data flow diagram is the *main ()* function, as mentioned earlier. Each data flow diagram at least has a process. A linked list of all these processes is maintained for each data flow diagram.

For each data flow diagram the processes are traversed.

1. Generate the local variables, if any.
2. Do the following for each process in the data flow diagram:
   (a) If there is no subprocess for that process, then the code is generated.
   (b) If there is a subprocess (which is another data flow diagram), then a function call with the parameters (if any) is generated. This function will be defined later. A prototype of this function is placed in the file *prototype.h*, which is automatically included in the application code at the beginning.

The data flow diagram preprocessor has been written in ANSI C.

7. A CASE STUDY

A case study is presented in this section. This case study deals with the working of a bookstore[1]. A subset of this problem is taken for the case study.

The first step in the problem is to identify the various objects. All the files in this application are the various objects. These objects are accounts, orders, customers, books, stocks and backorders. The next step is to convert any decision tables to programs in C++. This is an optional step, as an application may not have any decision tables at all. This application has two decision tables. The last step is to generate the application. The data flow diagrams shown in Fig. 6 are drawn using the GUI. Once all the details are entered, the application code is generated. The interconnection diagram drawn using the GUI is captured in the data structure shown in Fig. 5. The algorithm given earlier is used to generate the code. The user defined functions are generated first in a file called *userfuncs.cpp*. This file is automatically

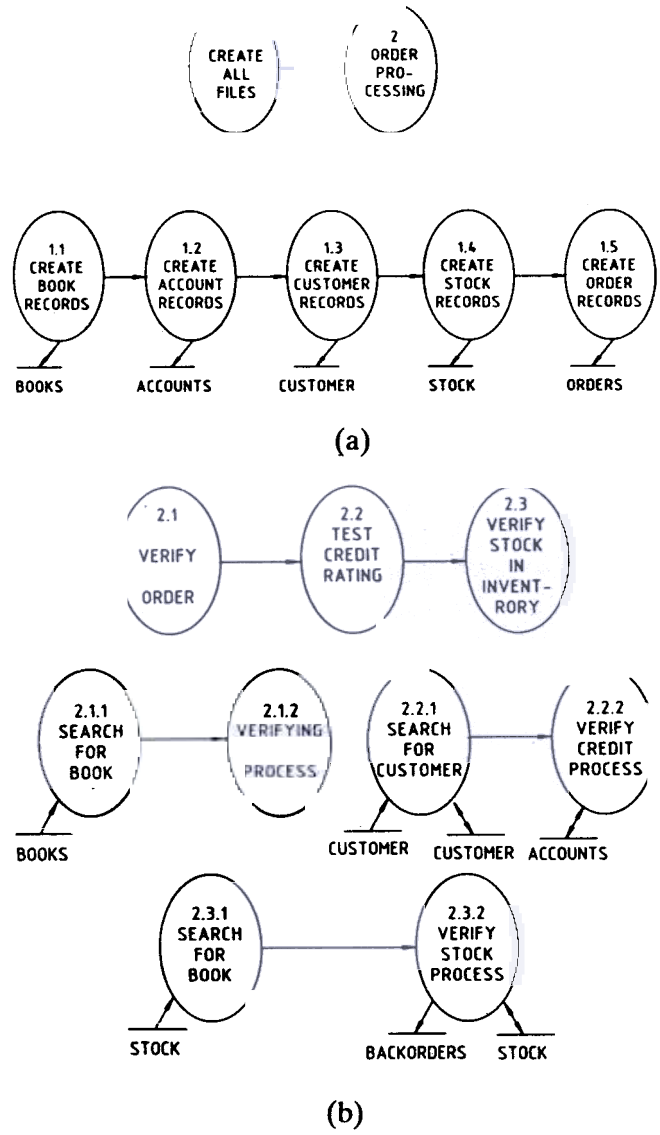

Figure 6. Data flow diagrams for the case study.

included in the final application code. Next the files for the objects are created. The next step is to generate the code for the processes. The code is generated in the file *main.cpp*.

8. CONCLUSIONS

In this paper a set of software tools are presented suited for business applications. The object-oriented programming was applied as the methodology for developing applications using these tools. These tools are non-procedural. The use of data flow diagrams and a graphical user interface to draw these diagrams makes the task of developing software easy for the end user who may be a novice. These tools are portable and easy

to use The source code generated is in C++, which can be modified by the user, if needed.

There is always scope for improvement in any work, as no work is perfect. The file object in the inheritance diagram contains a read-only file object, a write-only file object and a read-write object. It was found that most business applications do both reading and writing on a file. This means that mainly the read-write object is used. Hence, it might be better to have only the read-write object and call it the file object. This has the advantage that the user knows only one file object and not three file objects.

Currently, the process code is entered in the C++ language. This means that the end user has to know C++. Designing a language that has a correspondence to C++, yet simple for the end user, would greatly enhance the utility of this tool.

REFERENCES

1 Gopikrishna, M. & Rajaraman, V. Data flow oriented software tools for business data processing. *Comput. Sci. Informatics*, 1985, **15** (2), 9-22.

2. Rajaraman, V. Analysis and design of information systems. Prentice-Hall of India, New Delhi, 1991.

3. Brad, J. Cox, Object oriented programming–An evolutionary approach. Addison-Wesley, Reading, 1986.

4. Rajaraman, V. Decision tables. *In* Encyclopedia of computer science and technology, Vol. 24, Supplement 9. Marcel Dekker, Inc., 1991. pp. 85-106.

5. Oliver Jones, Introduction to the X window system. Prentice Hall, Englewood Cliffs, 1989.

6. Eric, F. Johnson & Reichard, Kevin. X window application programming. BPB Publications, New Delhi, 1990.