International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Novel Approach to Abstract the Data Flow Diagram from Java Application Program

Dr. R. N. Kulkarni[1], Mr. P. Pani Rama Prasad*[2]

Submitted: 27/04/2023 **Revised**: 28/06/2023 **Accepted**: 08/07/2023

Abstract: During the years 1920 to 1930, business processes came into use and they used flow charts for documenting the process flow. Further, it was developed as a flow chart to plan flow of information in computer programs. During 1960's onwards the designer of computer programs used algorithms and flow charts for the representation of the software design. In the year 1970, the Data Flow Diagram (DFD) was introduced, which depict the structured analysis and the data modeling. In this paper, a methodology is proposed and a tool is developed that is used to abstract complete DFD from given java program. The proposed abstraction is carried out with the help of following steps. First step, restructure the input java program that is amenable for the abstraction of DFD components. Second step, Identifying the input attributes set and output attributes set, data flow, different process, entities and data store. Third step, representing the abstracted components in the table format. Next step, redesigning the data flow diagram from the input DFD table.

Keywords: *Restructuring, DFD, entity, data store, process, data flow, java program.*

1. Introduction

The Computers science field is the fastest emerging area across the world. In particular, the Information Technology Industry is producing various types of application or system software's which are useful for numerous business organization and society across. For development of software application, the design is the most essential thing. Further, the design is realized with help of a suitable programming language. The Data flow diagram (DFD) is the important design diagram to get high level design of the java application. DFD is an independent design diagram and it is not a part of set of Unified Modeling Diagrams (UML). DFD is a combination of external sources, process, data inputs / outputs and data stores. DFDs are constructed with help of standard notations and symbols to represent flow of information between different components of the system.

In this paper, a new methodology is proposed and an automated tool is developed for the abstraction of design information from java program and further it is represented the in a table form called as Data Flow Table [DFT].

2. Related Work

The paper [22], discuss about the use of DFD to assure dependability. The article also explains the derivation of D-case from DFD and discuss various aspects of this method.

The authors in paper [17] discuss about the data flow diagram which signifies the relationships between the process and other system components used in the software application. The same concept is used in identifying relation between process in java program. The authors in paper [12], object-oriented data flow diagrams which as four components such external entities, processes, data stores, and classes. This work is useful in understanding the DFD components and its usages. In paper [10], author comprehensively discuss about the major deficiencies such as security concepts, abstraction levels, data elements, and deployment information in case of security threat modeling. This work gives a comprehensive view of the DFD in design and development of java application.

In paper [9], authors discuss about a hybrid diagram called as ER-Flow which provides an integrated view of data and source code with in the application. This work is useful in knowing the concept of data flow across different entities. Restructuring of a java program consists of multiple steps and are explained in paper [11]. These steps are used in our tool development. In paper [7], the author discussed about the control flow model and data flow models along with a brief comparison and discussion of their advantages and drawbacks of each model. The work specified here is useful in representation of data flow table. The authors in paper [4], discuss about organization of the processors which is based on whether they use the data flow execution model or in control flow model inside the system. This study is applied to identify the process inside the java application system. In this paper [3], authors discuss about the new CDESF tool which can manage the multiple data flows in the given application. This work given java in identifying data flow in a process for the given java application. This article [5] provides a suitable information for visualization

[1] *Prof & Head, Department of Computer Science and Engineering, Ballari Institute of Technology and Management, Ballari*
ORCID ID : 0000-0002-9948-1398
[2] *Asst. Prof, Department of Computer Science and Engineering, Ballari Institute of Technology and Management, Ballari*
ORCID ID : 0000-0002-5129-1309
[2]* *Corresponding Author Email: phanirama75@bitm.edu.in*

and modeling of input Java programs used for the static analysis, dynamic analysis, debugging of software and also maintenance of software application system. The work is used for understanding of data flow inside the java application.

In papers [16, 26], authors discussed about refinement for both DFDs and Privacy-Aware data flow diagrams as a distinct variety of structure - preserving map. This work is useful in comprehend with various artifacts of DFD and is used in our methodology. In paper [6], the authors proposed a procedure to abstract the program behavior and represent the same as a DFD, which involves multiple steps. This method is applicable to our methodology in creating of DFD from the given java application. In paper [18], discuss about structure and model for Java application to comprehend with the control as well as data flow analysis of execution of java application. This methodology discussed here is valuable in identifying entities and process in a java application. In paper [8], authors designed a visualization tool to extract data or control analysis from of java program. The output generated from the tool is useful in knowing data flow inside the java application. In this paper [1], authors developed a tool to draw DFD using Natural Language Interface (NLI) which is used to allow the consumer to create a query and classify the system functionality and limitations present in the system. This work is used to identify the process present inside the system. In paper [2], a novel tool developed to extract DFDs from the micro services code (java) using parsing technique and presented in table form.

In paper [13], Monolithic architecture technique is discussed, which combines the small independent micro services using the data flow driven approach to obtain the unique structure. The paper [19], a practical approach to design the system using a C++ program with help of DFD and Entity Relationship diagram are discussed in detail. In paper [27] tool is developed that extracts design of data flow from the given java code. The reverse engineering procedures are applied to identify different levels in the DFDs. The paper [25], talks about Data Flow Sequences (DFS) which is similar to the real life systems application development. It proposes a tool using an extended Markup Language for DFS implementation.

The paper [15] discuss about an object-oriented data flow diagrams using greedy approach to calculate the similarity score of class diagram verses DFD and also compare their different artifacts. The comparison between DFD and use case diagram is done to find the strengths and weakness of each diagram. Based on the study the author in paper [21] suggested to include the data flow in object- oriented approach. In paper [14], for construction and engineering problems data flow diagrams are used. A web based tool is developed to design prototype for engineering problems using data flow diagram. The paper [20] discuss about an App, where user will be knowing the different stages in which the software development activities are carried out using the App. The DFDs are useful to the user to understand the different steps in developing the application. The paper [24] focuses on problems where the hardware design and software design combined and it is called as hardware/software code sign. This code sign can be understood with the help of DFDs.

The paper [23] discuss about a smart hospital where ontological description knowledge data base of the hospital is maintained in ubiquitous computing environment using UML and XML technology. The DFD are used to comprehend with the high level design of the hospital system.

In paper [28] authors presenting a Predictive Object Point (POP) software which is useful to the software designers in planning the high quality software. With the aid of DFD, the software quality attributes are tested for completeness and correctness, The POP tool is used to assess the quality of OO software.

3. Methodology

The methodology proposed here is explained with help of a block diagram and it is as shown in Fig 3.1, which consists of the different components like restructuring, abstraction and representing the output in the three column table format. The methodology proposed is tested on different java programs with varying lines of code and results are tabulated in table 4.1. The results shows that the restructured programs are more efficient,

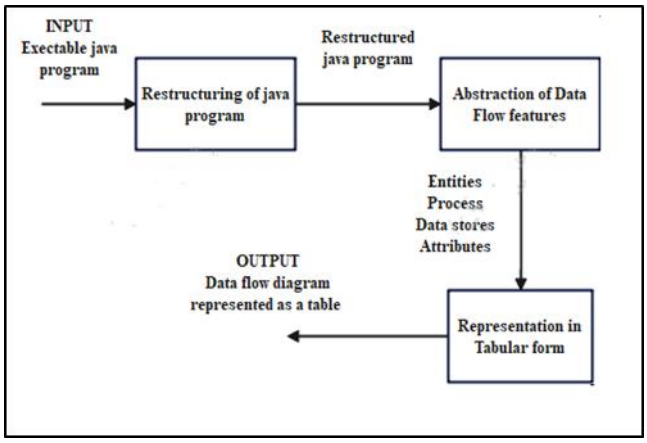


Fig 3.1 Block diagram to generate Data Flow Table

Initially, the executable java program is taken as an input for restructuring process. The basic purpose of restructuring the program is to remove the unnecessary things from the program without changing the its functionality. The restructured program is used for abstracting the different components of DFD such as process, entities, data store and data flows. Further, the abstracted components are stored and represented in a tabular format. The detailed description

is given below:

3.1. Restructuring of program.

3.2. Abstraction of Data flow components.

3.3. Generate a three column data flow table.

3.1. Restructuring of program.

Restructuring of a program is the process of making changes to the program without altering its functionality. The restructuring process involves the following steps:

(i) Reading the executable java program.

(ii) Eliminate the blank lines and comment lines.

(iii) Transforms multi statement lines to single statement line and multiline statements to single statement.

(iv) Assign physical line number to each statement.

3.2. Representation of data flow table

In this step, the restructured java program is taken as a input, further, the components required for the design of DFD such as entities, data flow, process and data store are abstracted and then stored in a text file.

3.3. Generate a three column data flow table

In this step, the abstracted features which are stored in the file is taken as input and then it is represented in a table format called as DFD table. The purpose of representing in tabular form is to easily generate the DFD from the table.

This section discuss about two algorithms first one, restructuring of java program. Second one, abstraction of DFD and representing the same in a three column tabular format. These two algorithms are as shown in fig 3.2.

[1. RESTRUCTURING OF INPUT JAVA PROGRAM]

INPUT: Java application program

OUTPUT: Restructured java application

Pre-Condition: Executable java program

Post-Condition: Restructured Executable java program

RESTRUCTURING JAVA PROG(Exam.java)

STR[1..n] ← Exam.java

if (! (Validate (STR [1..n]==NULL))) then

return InValid java

else

step 1.1: - [Remove comment Lines]

initialize comment=false

while (STR[1..n]!= NULL) do

if (STR.contains("/*")) then

comment = true

end if

if(lSTR.contains("*/")) then

comment = false

end if

if(STR.contains("//")) then

comment = true

end if

if(!comment) then

printl(STR)

end if

end while

step 1.2: - [Remove Blank Lines]

Initialize NEW_LINE='\n'

while (STR[1..n]!= NULL) do

if (!STR.isEmpty()) then

writer_to_file(STR);

writer_to_file(NEW_LINE)

end if

end while

step 1.3: - [Convert the multiple statements into single line]

while (STR[1..n]!= NULL) do

STR = STR.replaceAll(";“ , "; \n")

writer_to_file(STR);

writer_to_file(NEW_LINE)

end while

step 1.4: - [Convert the multiline statements into single line]

while (STR[1..n]!= NULL) do

while (STR[1..n]!=';)

STR = STR1+STR.replaceAll(";“ , "; \n")

end while

writer_to_file(STR);

writer_to_file(NEW_LINE)

end while

step 1.5: - [Insert Line No to all the statements in the program]

initialize line no=1

STR[1..n] ← Exam.java

while (STR.nextToken()) do

initialize String name = STR.nextLine()

print(lineno+name)

lineno = lineno + 1

end while

end if

[2. ABSTRACTION OF DFD COMPONENTS]

INPUT: Restructured Java application program

OUTPUT: Data flow table in tabular format

Pre-Condition: Restructured Executable java program

Post-Condition: Tabular representation of data flow diagram.

DFD_JAVA_PROG(Exam.java)

ST[1..n] ← STR[1..N]

if (! (Validate (ST [1..n]==NULL))) then

return InValid Restructered java program

else

step 2.1: - [Identify Entities]

while (ST[1..n]!= NULL) do

if (!ST[1..n]== *Object*) then

E_{entity}[1..n] ← Object.name

end if

end while

step 2.2:- [Identify Process]

while (ST[1..n]!= NULL) do

if (!ST[1..n]== *function*) then

$P_{process}$[1..n] ← function .name

else

$A_{attribites}$[1..n] ← {Attributes}

end if

end while

step 2.3: - [Identify Data store]

while (ST[1..n]!= NULL) do

if (ST[1..n]== "string query = "select * from ")

then

D_{store}[1..n] ← DB .name

end if

end while

[3. REPRESENTING DFD COMPONENTS IN A TABLE FORM]

while (ST[1..n]!= NULL) do

if (ST[1..n]==E_{entity}[1..n])

SOURCE[1..n] ← E_{entity}[1..n])

end if

if (ST[1..n]==$P_{process}$[1..n])

DESTINATION[1..n] ← $P_{processes}$[1..n])

end if

if (ST[1..n]==D_{store}[1..n])

DESTINATION[1..n] ← D_{store}[1..n])

end if

if (ST[1..n]==$A_{attributes}$[1..n])

DATAFLOW[1..n] ← $A_{attributes}$[1..n])

end if

end while.

[4. REDESIGING THE DFD]

while (DATA_TABLE[1..n]!= NULL) do

if (DATA_TABLE[1..n]== E_{entity}[1..n])

draw_object (E_{entity}[1..n])

end if

if (DATA_TABLE[1..n]== $P_{processes}$[1..n])

draw_object ($P_{processes}$[1..n])

end if

if (DATA_TABLE[1..n]== D_{store}[1..n])

draw_object (D_{store}[1..n])

end if

if (DATA_TABLE[1..n]== $A_{attributes}$[1..n]

draw_object ($A_{attributes}$[1..n])

end if

end while

Figure 3.1 Algorithm for restructuring and abstraction of data components.

Fig 3.1 Algorithm for restructuring and abstraction of data components

4. Results and Discussion

When the restructuring process is used, the time taken to execute the program is reduced. The comparison between the java programs before and after the restructuring is as

given in table 4.1. The table consists of four columns where first, second columns are Name of the program and number of lines of code. Third and fourth columns are execution time in milli-seconds(ms), before and after restructuring the java program.

Tab 4.1 Execution Time Comparison Table

SN	Name of the Program	No. of Lines of Code	Time taken to execute	
			Before Restructuring	After Restructuring
1.	emp.java	82	43	36
2.	stud.java	56	37	32
3.	admissions.java	349	95	84
4.	atm.java	303	89	73
5.	placement.java	183	61	54

The corresponding graph of Execution time of Normal Vs Restructured program is as shown in figure 4.2. The graph indicates that time taken to execute the restructured java program is less when it is compared with execution time of normal program without restructuring.

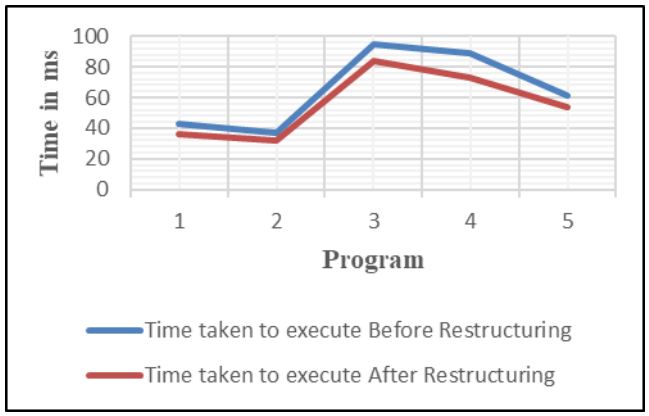


Fig 4.2 Execution Time Comparison graph

The proposed automated tool restructures the input executable program shown in fig 4.3 and the resultant restructured program is as shown in fig 4.4. Further the components required for the design of DFD are abstracted and stored in the text file. The tool reads this file and then the data flow diagram is drawn. To draw the diagram an intermediate available tool is used.

```
//Student class
class Student {
String USN, name, mailed;
int sem, mobile_no;
student (String USN, String name, String mailed, int sem, int mobile_no) {
  this.USN = USN;
  this.name = name;
  this.mailid = mailid;
  this.sem = sem;
  this.mobile_no = mobile_no;
}
void fillForm (String USN, String name, String mailed, int sem, int mobile_no) {
  int n;
  char sub[10];
  double fees;
  Scanner in = new Scanner(System.in);

  System.out.println("Enter No of subjects:-");
  n = in,nextInt();
  System.out.println("Enter subject codes:-");
  For ( i=0; i<n; i++)
      sub[i] = in.nextLine();
  System.out.println("Enter Total fees:-");
  fees = in,nextDouble();
}
/* Payment function */
int payFees(int n, string sub[], double fees) {
  String UPI, UTR=0; System.out.println("Enter UPI No:-");
  UPI = in,nextLine();
  System.out.println("Total fess:-"+fees);
  System.out.println("Enter UTR No:-");UTR = in,nextLine();
  if(UTR > 0)
    System.out.println(" Payment is successful");
  else
    System.out.println(" Payment is unsuccessful");
return UTR;
}

}
```

```
class Proctor {
string pname, app_no;
Proctor(steing pname) {
  this.pname = pname;
}
void PfillOnline() {
 String UTR;
 fillForm (USN, name, mailed,  sem, mobile_no);
 UTR = payFees(n, sub[], fees);
 System.out.println("Enter Exam application No.:-");
 app_no = in,nextLINE();
 string query = "select * from student";
 class.forName("com.mysql.cj.jdbc.Driver");
 Connection con = DriverManager,getConnection(USN, name, sem, n, sub, app_no, UTR);
 System.out.println("Connection established successfully");
 Statement st = con.createStatment();

 ResultSet rs = st.executeQuery(query);
 st.close();
 con. close();
 System.out.println("Data base Connection closed");
}
void printApplication(){
 onlineexamfill();
 string query = "select * from student";
 class.forName("com.mysql.cj.jdbc.Driver");

 Connection con = DriverManager,getConnection(USN, name, sem, n, sub, app_no, UTR);
 System.out.println("Connection established successfully");
 Statement st = con.createStatment();
 ResultSet rs = st.executeQuery(query);
 rs.next();
 USN = rs.getString("USN");
 name = rs.getString("name"); sem = rs.getString("sem");
 UTR = rs.getString("UTR"); app_no = rs.getString("app_no");
 st.close();
 con. close();
}
}
//Main program
Class Test1 {
public static void main (String args[]) {
 Student st = new Student("3BR21CS001", "shayam", shyam@bitm.edu.in, 5, 8443165674);
 Proctor pt = new Proctor("phani rama");
 st. fillForm();
 st. payFees(8,sub, 1700);
 pt. printApplication();
 }
}
```

Fig 4.3 Input Executable Java program

After the restructuring process the restructured java program of student examination online filling application as shown in figure 4.4.

```
1.  class Student {
2.  String USN, name, mailed;
3.  int sem, mobile_no;
4.  student (String USN, String name, String mailed, int sem, int mobile_no) {
5.    this.USN = USN;
6.    this.name = name;
7.    this.mailid = mailid;
8.    this.sem = sem;
9.    this.mobile_no = mobile_no;
10. }
11. void fillForm (String USN, String name, String mailed, int sem, int mobile_no) {
12.   int n;
13.   char sub[10];
14.   double fees;
15.   Scanner in = new Scanner(System.in);
16.   System.out.println("Enter No of subjects:-");
17.   n = in,nextInt();
```

```
18.  System.out.println("Enter subject codes:-");
19.  For ( i=0; i<n; i++)
20.       sub[i] = in.nextLine();
21.  System.out.println("Enter Total fees:-");
22.  fees = in,nextDouble();
23. }
24. int payFees(int n, string sub[], double fees) {
25.  String UPI, UTR=0;
26.  System.out.println("Enter UPI No:-");
27.  UPI = in,nextLine();
28.  System.out.println("Total fess:-"+fees);
29.  System.out.println("Enter UTR No:-");
30.  UTR = in,nextLine();
31.  if(UTR > 0)
32.    System.out.println(" Payment is successful");
33.  else
34.    System.out.println(" Payment is unsuccessful");
35. return UTR;
36. }
37. }
38. class Proctor {
39. string pname, app_no;
40. Proctor(steing pname) {
41.   this.pname = pname;
42. }
43. void PfillOnline() {
44.  String UTR;
45.  fillForm (USN, name, mailed,  sem, mobile_no);
46.  UTR = payFees(n, sub[], fees);
47.  System.out.println("Enter Exam application No.:-");
48.  app_no = in,nextLINE();
49.  string query = "select * from student";
50.  class.forName("com.mysql.cj.jdbc.Driver");
51.  Connection con = DriverManager,getConnection(USN, name, sem, n, sub, app_no, UTR);
52.  System.out.println("Connection established successfully");
53.  Statement st = con.createStatment();
54.  ResultSet rs = st.executeQuery(query);
55.  st.close();
56.  con. close();
57.  System.out.println("Data base Connection closed");
58. }
59. void printApplication(){
60.  onlineexamfill();
61.  string query = "select * from student";
62.  class.forName("com.mysql.cj.jdbc.Driver");
63.  Connection con = DriverManager,getConnection(USN, name, sem, n, sub, app_no, UTR);
64.  System.out.println("Connectionestablished");
65.  Statement st = con.createStatment();
66.  ResultSet rs = st.executeQuery(query);
67.  rs.next();
68.  USN = rs.getString("USN");
69.  name = rs.getString("name");
70.  sem = rs.getString("sem");
71.  UTR = rs.getString("UTR");
72.  app_no  = rs.getString("app_no");
73.  st.close();
74.  con. close();
75. }
76. }
77. Class Test1 {
78. public static void main (String args[]) {
79.  Student st = new Student("3BR21CS001", "shayam", shyam@bitm.edu.in, 5, 8443165674);
80.  Proctor pt = new Proctor("phani rama");
81.  st. fillForm();
82.  st. payFees(8, sub, 1700);
83.  pt. printApplication ();
84.  }
85. }
```

Fig 4.4 Restructured Java program

The restructured java program as shown in figure 4.2. In this

program, *Student and Proctor* classes are the external entities. This java program has several process such as *fillForm, payFees ,fillOnline and printApplication*. The program has one data store *student*, which maintains student examination application details.

The table 4.5 depicts the data flow inside the given java program. The data flow table is represented using the thee columns source, destination and data flow. In this table source or destination may be an entity, process or data store. Data flow indicates the data that flows between source and destination. Here data dataflow represents the attributes which are either input or output from the entity or a process or a database. From table 4.1, the first row indicates Student entity fills the examination form using formFill process. In this interaction data flows between entity and process is the student details such as USN, name, mobile_no, mailed, semester, number of subjects n, subject code, total fees to be paid. In second row, formFill process interacts with payFees process to pay the examination fees using UPI, fees and UTR attributes as a data flow. In case of third, fourth and fifth rows, fillOnline process communicates with Proctor entity, fillform and payFees process to make online entry using attributes USN, name, n, sub, fees, UPI, UTR and application number app_no. Once online entry of student examinations details is made, same details are kept inside the student database. The similar type of information is depicted in sixth row of the data flow table. Here the fillOnline process interacts with student data store. In case of seventh and eight rows, the student examination application print out is given to Student entity. This step involves interaction between student data store, printAppliaction process and Student entity respectively.

Table 4.5. Data flow table

SOURCE	DESTINATION	DATAFLOW
Student	fillForm	USN, name, mobile_no, mailed, sem, n, sub, fees
fillForm	payFees	USN, name, sem, n, sub, UPI, UTR, fees
payFees	fillOnline	UPI, UTR, fees
fillForm	fillOnline	USN, name, sem, n, sub, UPI, UTR, fees
Proctor	fillOnline	pname
fillOnline	student	USN, name, mobile_no, mailed, n, sub, fees, UTR, app_no
student	printApplication	USN, name, mobile_no, mailed, n, sub, fees, UTR, app_no
printApplication	Student	USN, name, mobile_no, mailed, n, sub, fees, UTR, app_no

A Data Flow Diagram consists of Processes (Ps), Data Flows (Df), Data Stores (St) and External Entities (Et). The data flow diagram shown in figure 4.7 is verified with the DFD rules for its completeness and correctness.

Let DFD be a data flow diagram, then

$DFD = \{Ps, Df, St, Et\}$ ----------(1)

where

Ps = {ps_1, ps_2, ps_3…, ps_n} is a finite set of processes;

Df = {df_1, df_2, df_3…, df_n} is a finite set of data flows;

St = {st_1, st_2, st_3….., st_n} is a finite set of data stores;

Et = {et_1, et_2, $et_{3\ldots}$…, et_n} is a finite set of external entities;

Equation (1) defines a Data flow diagram, which is a set of consists of a set of processes, data flows, data stores and external entities.

Let *Cd* be a context diagram then

$Cd = \{< eti, dfj, ps1 >, < p1, dfk, eti >\}$

$j \neq$ k 1, ≤ i, j, k ≤ n----------(2)

In equation (2), Context diagram (Cd) consists of only one process along with the set of external entities (et_i) and data flows(df_i). Data flow can be connected from external entity to a process and vice versa but the data flow must be a different data flow. Note that, data store can only exist in data flow diagram but not context diagram. From the context level diagram as shown in fig 4.6, there is only one process named as *onlineExamFill*. The external entities named as *Student* and *Proctor*.

$\forall st_i, st_j \in St, st_i \neq st_j$ 1, ≤ i, j ≤ n ---------- (3)

From equation (3) the two process in a DFD should not have same name. Duplicate process names are not allowed. The same rules apply for data flows. The process names represented as a set Ps = {*fillForm, payFees ,fillOnline and printApplication}* which are distinct and unique.

The same rules are applied for data flows shown in equation (4), data stores in equation (5) and external entities shown in equation (6).

$\forall sti, stj \in St, sti \neq stj$ 1, ≤ $i, j \leq n$ ---------- (4)

The data flows (Df) are the attributes passed between the process, the set of attributes are Df = {*USN, name, mobile_no, mailed, sem, n, sub, fees}* are unique and different.

$\forall sti, stj \in St, sti \neq stj$ 1, ≤ $i, j \leq n$ ---------- (5)

There is only one data store (St) in the given java program

St = {student}, which is the student data base and it is by default unique only.

$\forall eti, etj \in Et, eti \neq etj 1, \leq i, j j \leq n$ ----------(6)

The java program has two entities Et = {Student, Proctor}, which are not duplicate and hence are unique and distinct.

$\forall eti, etj, sti, stj, dfk \in DFD, then\ DFD \neq \{eti, dfk, et >\}\ and\ DFD \neq \{< sti, dfk, stj >\}, 1 \leq i, j, k \leq n$ ------- (7)

Equation (7) indicates that for any data flow diagram, a data flow cannot connect from one external entity to another external entity and a data flow cannot also connect from one data store to another data store. From the data flow table 4.5, External entities(Et) *Student and Proctor* are not directly connected. There only one Data store(St) *student*, hence directly connecting with another entity does not arise.

$\forall eti, stj, stj, dfk \in DFD, then\ DFD \neq \{< eti, dfk, >\}\ and\ DFD \neq \{< stj, dfk, eti >\}, 1 \leq i, j, k \leq n$ ----------(8)

Equation (8) indicates that for any data flow diagram, m, a data flow cannot connect from one external entity to data store and a data flow cannot also connect from one data store to external entity. From the data flow table 4.5, external entities(Et) *Student and Proctor* are not directly connected to the data store(St) *student* and vice versa is also not directly connected.

Consider the context level diagram for the given java program which is as shown in fig 4.6. The *onlineExamfill* process interacts with *Student* entity for collecting student data and *Proctor* Entity is used to fill the student data in online mode.

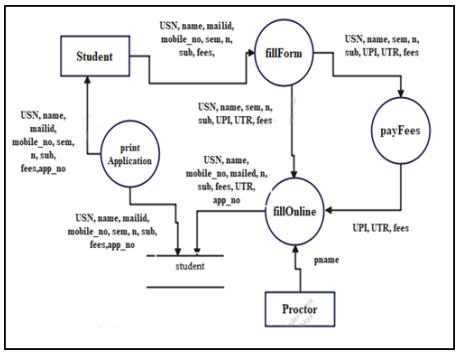


Fig 4.6 context level DFD for the given java program

The Level-1 DFD which is as shown in figure 4.7 will be generated with the help of data flow table which is as shown in table 4.5. In this Level-1 DFD, *Student* and *Proctor* are the external entities. There are four process in the DFD such as *fillForm*, *payFees*, *fillOnline and printApplication* along with single data store called as student.

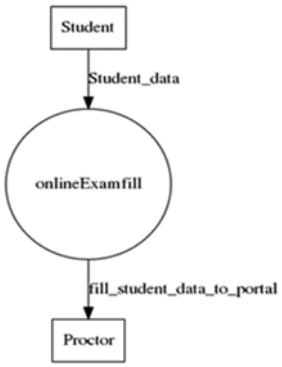


Fig 4.7 DFD for the given java application

The DFD may be drawn using the plantUML generic tool. The data flow table as shown in table 4.5, consists of source, destination and data flow, by using this information plantUML code is generated. The plantUML code to draw context level diagram and Level-1 DFD is as shown in fig. 4.8 and fig 4.9 respectively.

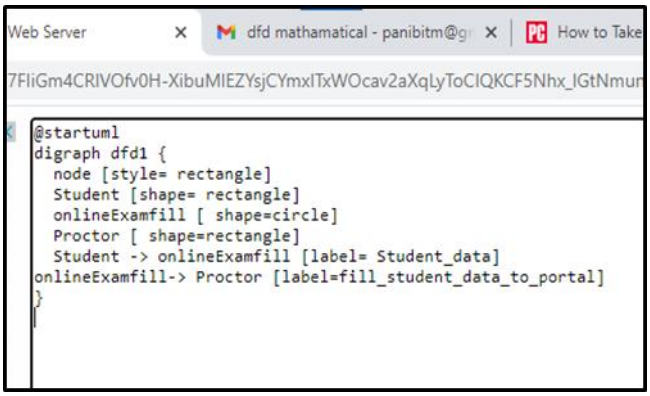


Fig 4.8. Code to draw the context level diagram

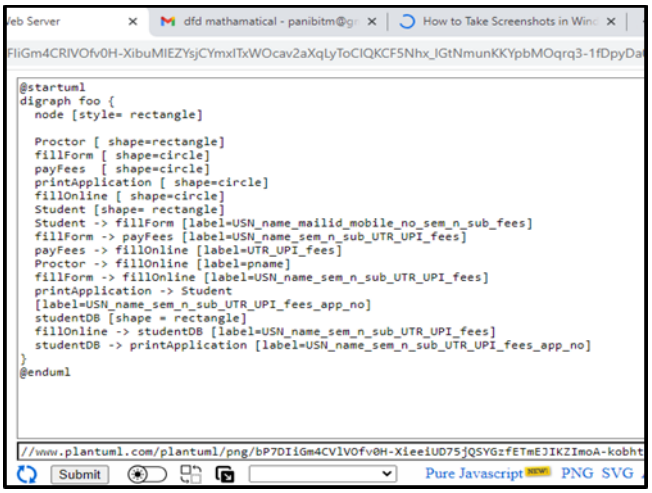
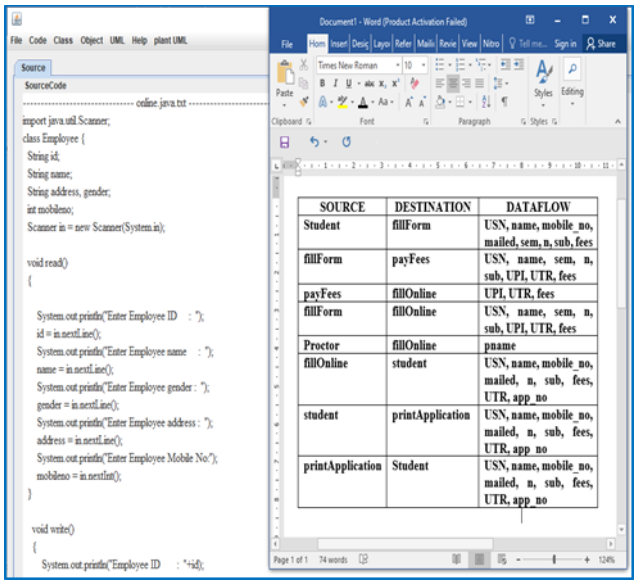


Fig 4.9. Code to draw the Level-1 DFD

The output generated form the tool is as shown in the fig 4.10. The restructured java program along with the data flow table are as shown in fig 4.10



SOURCE	DESTINATION	DATAFLOW
Student	fillForm	USN, name, mobile_no, mailed, sem, n, sub, fees
fillForm	payFees	USN, name, sem, n, sub, UPI, UTR, fees
payFees	fillOnline	UPI, UTR, fees
fillForm	fillOnline	USN, name, sem, n, sub, UPI, UTR, fees
Proctor	fillOnline	pname
fillOnline	student	USN, name, mobile_no, mailed, n, sub, fees, UTR, app_no
student	printApplication	USN, name, mobile_no, mailed, n, sub, fees, UTR, app_no
printApplication	Student	USN, name, mobile_no, mailed, n, sub, fees, UTR, app_no

Fig 4.10 Output Data Flow Table

5. Conclusion

In this paper, an attempt is made to abstract DFD from the given java program. The tool proposed here first restructures the given java application then abstracts the elements such as entity, data store, process, data flow and attributes from the java program. The same information is represented in table format which has three columns. Further, DFD is redesigned by using the data table. The tool is tested for its correctness and completeness by taking different types of java programs.

References

[1] Sehrish Munawar Cheema, Saman Tariq, Ivan Miguel Pires, “A natural language interface for automatic generation of data flow diagram using web extraction” techniques,Journal of King Saud University - Computer and Information Sciences, Vol 35, no 2,2023.

[2] Simon Schneider, Riccardo Scandariato, “Automatic extraction of security-rich dataflow diagrams for microservice applications written in Java”, Journal of Systems and Software,Vol 202, 2023.

[3] Paolo Ceravolo, Ernesto Damiani, Emilio Francesco Schepis, Gabriel Marques Tavares, “Real-time probing of control-flow and data-flow in event logs”, Procedia Computer Science, Vol 97, 2022.

[4] M. B. Khan, A. R. Khan and H. Alkahtani, "Exploring the approaches to data flow computing," Computers, Materials & Continua, vol. 71, no.2, 2022.

[5] Stephan Seifermann, Robert Heinrich, Dominik Werle, Ralf Reussner, “Detecting violations of access control and information flow policies in data flow diagrams”, Journal of Systems and Software, Vol 184, 2022.

[6] Alshareef, Hanaa & Stucki, Sandro & Schneider, Gerardo. (2021). “Refining Privacy Aware Data Flow Diagrams”, Software Engineering and Formal Methods, 19th International Conference, SEFM -2021, 2021.

[7] Lang Feng, Jiayi Huang, Jeff Huang, and Jiang Hu. 2021.” Toward Taming the Overhead Monster for DataFlow Integrity”. ACM Trans. Des. Autom. Electron. Syst. Vol 27, no. 3, 2021.

[8] Faily, S., Scandariato, R., Shostack, A., Sion, L., Ki-Aries, D. “Contextualisation of Data Flow Diagrams for Security Analysis”, Lecture Notes in Computer Science, vol 12419. Springer, Cham. 2020.

[9] C. Ordonez, S. Tahsin Al-Amin and L. Bellatreche, "An ER-Flow Diagram for Big Data," IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 2020.

[10] Laurens Sion, Koen Yskout, Dimitri Van Landuyt, Alexander van den Berghe, and Wouter Joosen. “Security Threat Modeling: Are Data Flow Diagrams Enough”, IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20). Association for Computing Machinery, New York, 2020.

[11] Dr. R.N. Kulkarni, P. Pani Rama Prasad, “Restructuring of Java Program to be amenable for Reengineering”, Journal of Engineering Science and Technology, vol 02 no 06, 2019.

[12] F. Irhamn and D. Siahaan, "Object-Oriented Data Flow Diagram Similarity Measurement Using Greedy Algorithm," 2019 1st International Conference on Cybernetics and Intelligent System (ICORIS), Denpasar, Indonesia, 2019.

[13] Shanshan Li, He Zhang, Zijia Jia, Zheng Li, Cheng Zhang, Jiaqi Li, Qiuya Gao, Jidong Ge, Zhihao Shan, “ A dataflow-driven approach to identifying micro-services from monolithic applications”, Journal of Systems and Software ,Vol 157, 2019.

I. V. Trubnikov, O. V. Minakova and O. V. Kuripta, "Framework for Building Data Flow Diagramm Based Applications," IEEE International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon), Vladivostok, Russia, 2019.

[14] F. Irhamn and D. Siahaan, "Object-Oriented Data Flow Diagram Similarity Measurement Using Greedy Algorithm," IEEE 1st International Conference on Cybernetics and Intelligent System (ICORIS), Denpasar, Indonesia, 2019.

[15] Bryan L. Guibijar, MSCS, “Data Flow Diagram (DFD)

in Developing Online Product Monitoring System (OPMS) of DTI", International Journal of Trend in Scientific Research and Development (IJTSRD) ISSN: 2456-6470, Vol 2, no 6, 2018.

[16] B. Yu and C. T. Silva, "VisFlow - Web-based Visualization Framework for Tabular Data with a Subset Flow Model," in IEEE Transactions on Visualization and Computer Graphics, vol. 23, Jan. 2017.

[17] Soomro, Safeeullah & Alansari, Zainab & Riyaz Belgaum, Mohammad. "Control and Data Flow Execution of Java Program. Asian Journal of Scientific Research, 2017.

[18] Mohammed H. S. Al Ashry,"Importance of Data Flow Diagrams and Entity Relationships Diagrams to Data Structures Systems Design in C++ :A Practical Example," Journal of Management and Strategy, Journal of Management and Strategy, Sciedu Press, vol. Vol 8, no 4, pp 51-61, 2017.

[19] Wulandari, Wati, and Albertus Dwi Yoga Widiantoro. "Design data flow diagram for supporting the user experience in applications." International Journal of the Computer, the Internet and Management Vol.25 No.2 pp. 14-20, 2017.

[20] Arwa Y. Aleryani, "Comparative Study between Data Flow Diagram and Use Case Diagram", International Journal of Scientific and Research Publications, Volume 6, Issue 3, March 2016.

[21] Mahmoud, M. S, "Development Of HealthCare System For Smart Hospital Using UML and XML Technology",International Journal of Intelligent Systems and Applications in Engineering, 2(3), 38–45, 2014

[22] N. Olayan, V. Patu, Y. Matsuno and S. Yamamoto, "A Dependability Assurance Method Based on Data Flow Diagram (DFD)," 2013 European Modelling Symposium, Manchester, UK, 2013, pp. 113-118, doi: 10.1109/EMS.2013.20.

[23] Schaumont, P.R. "Analysis of Control Flow and Data Flow. In: A Practical Introduction to Hardware/Software", Codesign. Springer, Boston, MA, 2013.

[24] James PH Coleman, "Data Flow Sequences: A Revision of Data Flow Diagrams for Modelling Applications using XML", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 4, No.5, 2013.

[25] Kulkarni, R.N., Aruna, T., Amrutha, N. "Abstraction of Design Information from Procedural Program", High Performance Architecture and Grid Computing. HPAGC 2011. Communications in Computer and Information Science, Springer, Berlin, Heidelberg, vol 169, 2011.

[26] Arun Lakhotia ,"An approach to recovering data flow oriented design of a software system", , IEEE Software, pages 74–81, Jan. 2000.

[27] Vijay Yadav, Raghuraj Singh, Vibhash Yadav, "Evaluation of OO Software Quality by Using Predictive Object Points (POP) Metric", Int J Intell Syst Appl Eng, vol. 11, no. 2s, pp. 328–336, Jan. 2023.

[28] Mrs. Monika Soni. (2015). Design and Analysis of Single Ended Low Noise Amplifier. International Journal of New Practices in Management and Engineering, 4(01), 01 - 06. Retrieved from http://ijnpme.org/index.php/IJNPME/article/view/33

[29] Goar, V. ., Yadav, N. S. ., & Yadav, P. S. . (2023). Conversational AI for Natural Language Processing: An Review of ChatGPT. International Journal on Recent and Innovation Trends in Computing and Communication, 11(3s), 109–117. https://doi.org/10.17762/ijritcc.v11i3s.6161