PSEUDOCODE

FUNCTIONS

- ChooseFile: Opens a fileChooser with extension as parameter, to be appended in the file name, returns Directory and file name
- This: This object
- ThisWindow: Current Window (Does not change with dialog boxes)
- GeneratePositionIn: Outputs positions of inflows
- GeneratePositionOut: Outputs positions of outflows

STRUCTURES

DATATYPE

Datatype of DD variable

- Name (STRING) : Variable name
- Type (ENUM) : Variable Type

DDBACKEND

DD Backend

- Data (JSON file) : Object properties
- Names (SET of DATATYPE): Maps dd units with structures

DECOMPOSEDATA

Decompose Data of Data Process

- DFDBackend (DFDBACKEND)
- DDBackend (DDBACKEND)

DFDBACKEND

DFD Backend

- Connects (MAP of TWOPATH to PATHS) : Connections beween two data entities
- Data (JSON file): Object properties
- UndirGraph (GRAPH of SHAPE): Undirected graph of shapes

EDGE

Edge

- u (POSITION) : First point
- v (POSITION) : Second point

FILE

File (Inherits WINDOW, DDBACKEND, IMAGE as per context)

INFLOW

External incoming dataflow (Combination of data process with incoming dataflow)

OUTFLOW

External outgoing dataflow (Combination of data process with outgoing dataflow)

PATH

Path

- Type (ENUM): SHAPE, TWOPATH

PATHS

Array of PATH

POSITION

Array of 2 INTEGER: x, y

POSITIONS

Set of 2 POSITION : Upper left, Lower right

SHAPE

Shape

- Name (STRING) : Name of Shape
- Type (SHAPETYPE) : Name of Shape Type
- Position (POSITIONS) : Position of shape
- InDataFlow (SET of SHAPES) : Set of incoming data flows (NULL for data flows)
- OutDataFlow (SET OF SHAPES) : Set of outgoing data flows (NULL for data flows)
- Decompose (DECOMPOSEDATA) : Decompose data (NOT NULL for data process)

SHAPETYPE

Shape Type: Enumeration of DataProcess, DataFlow, DataStore, ExternalEntity, ExternalOutput

SHAPEANCHOR

Anchor point of Shape

- Shape (SHAPE): Associated Shape
- Anchor (ENUM [UP, DOWN, LEFT, RIGHT]): Anchor Point Position

TWOPATH

Path of length 2

- Uedge (SHAPE) : Edge outgoing from data entry

- Vedge (SHAPE) : Edge incoming to data entry

WINDOW

Window

- DDBackend (DDBACKEND) : DD Backend
- DFDBackend (DFDBACKEND) : DFD Backend
- Dir (STRING) : Directory
- IsChanged (BOOLEAN) : Check if the file is changed since last save
- Title (STRING) : File Name (Default: “Untitled”)
- Workspace (INTERACTIVEIMAGE) : Workspace
- Log (PANE) : Debugging Pane
- Parent: Parent Window (Default: NULL)

ALGORITHMS

DDWIZARD

```
1.  ALGORITHM DDWizard(INTEGER checkCases) : VOID {
2.      OPEN DDWizard dialog box AS dw;
3.      okDisabled := TRUE;
4.      ROLLBACK IF (dw.cancelPressed()) {
5.          INPUT dw.Name;
6.          IF (dw.Name IN ThisWindow.DDBackend.Names) {
7.              IF (checkCases = 0) {
8.                  errorNameExists();
9.                  GOTO 5;
10.             } ELSE IF (checkCases = 1) {
11.                 n := This.End.Names;
12.                 IF (dw.Name IN n) {
13.                     This.Name := dw.Name;
14.                     This.Type := This.End.Names[dw.Name].Type;
15.                 } ELSE {
16.                     errorNameExists();
17.                     GOTO 5;
18.                 }
19.             } ELSE {
20.                 n := This.Start.Names;
21.                 IF (dw.Name IN n) {
22.                     This.Name := dw.Name;
23.                     This.Type := This.Start.Names[dw.Name].Type;
24.                 } ELSE {
25.                     errorNameExists();
26.                     GOTO 5;
27.                 }
28.             }
29.         } ELSE {
30.             INPUT dw.Type;
31.             IF (dw.Type = Struct) CreateStruct(dw.Name);
32.             This.Name := dw.Name;
```

```
33.                             This.Type := dw.Type;
34.                             ADD dw.Name IN ThisWindow.DDBackend.Names;
35.                             IF (checkCases = 1) ADD dw.Name IN This.End.Names;
36.                             ELSE IF (checkCases = 2) ADD dw.Name IN This.Start.Names;
37.                     }
38.                     okDisabled := FALSE;
39.                     WHILE (NOT dw.okPressed());
40.             }
41.             CLOSE dw;
42.     }
```

DEBUGFILE

```
1.      ALGORITHM DebugFile() : BOOLEAN Ans {
2.              du := DebugUnnecessary();
3.              di := DebugIsolated();
4.              db := DebugBalanced();
5.              Ans := du OR di OR db;
6.      }
```

DEBUGBALANCED

```
1.      ALGORITHM DebugBalanced() : BOOLEAN Ans {
2.              Ans := FALSE;
3.              FOR DATAPROCESS dp IN ThisWindow.DFDBackend.Data {
4.                      Ans := Ans OR s.DebugBalancedShape();
5.              }
6.      }
```

DEBUGBALANCEDSHAPE

```
1.      ALGORITHM DebugBalancedShape() : BOOLEAN Ans {
2.              Ans := FALSE;
3.              IF (This.Decompose NOT NULL) {
4.                      IF (
5.                              (
6.                                      SET(
7.                                              FOR SHAPE s2 IN This.InDataFlow
8.                                      ) != SET(
9.                                              FOR INFLOW if IN This.Decompose.DFDBackend.Data
10.                                     )
11.                             ) OR
12.                             (
13.                                     SET(
14.                                             FOR SHAPE s2 IN This.OutDataFlow
15.                                     ) != SET(
16.                                             FOR (
17.                                                     OUTFLOW of
18.                                             ) IN This.Decompose.DFDBackend.Data
19.                                     )
20.                             )
21.                     ) {
22.                             PRINT “Balancing Error in “ + This.Name + “\n” IN ThisWindow.Log;
23.                             Ans := TRUE;
24.                     }
```

```
25.             }
26.     }
```

DEBUGISOLATED

```
1.      ALGORITHM DebugIsolated() : BOOLEAN Ans {
2.              g := ThisWindow.DFDBackend.UndirGraph;
3.              n := g.Nodes.length;
4.              IF (n == 0) Ans := FALSE; ELSE {
5.                      isVisited := {s: FALSE FOR SHAPE s IN g};
6.                      s0 := RANDOMCHOICE(g.Nodes);
7.                      isVisited[s0] := TRUE;
8.                      QUEUE q;
9.                      q.Enqueue(0);
10.                     REPEAT UNTIL (q.IsEmpty()) {
11.                             u := q.Dequeue();
12.                             isVisited[u] := TRUE;
13.                             FOR EDGE e FROM u {IF (NOT isVisited[e.v]) q.Enqueue(e.v);)
14.                     }
15.                     FOR SHAPE s, BOOLEAN b IN ENUMERATE(isVisited) {
16.                             IF NOT b {
17.                                     PRINT (
18.                                             “Isolation Error in “ + s.Name + “\n”
19.                                     ) IN ThisWindow.Log;
20.                                     Ans := TRUE;
21.                             }
22.                     }
23.             }
24.     }
```

DEBUGUNNECESSARY

```
1.      ALGORITHM DebugUnnecessary() : BOOLEAN Ans {
2.              mc := ThisWindow.DFDBackend.Connects;
3.              FOR TWOPATH x IN mc.keys() {
4.                      IF (mc[x].length() > 1) {
5.                              PRINT (“Unnecessary flow in” + mc + “\n”) IN ThisWindow.Log;
6.                              Ans := TRUE;
7.                      }
8.              }
9.      }
```

DECOMPOSE

```
1.      ALGORITHM Decompose() : VOID {
2.              OPEN NEW DECOMPOSEWINDOW as dw;
3.              dw.Parent := ThisWindow;
4.              ThisWindow := dw;
5.              ThisWindow.Title := This.Name + “_Decompose”;
6.              IF (This.Decompose NULL) {
7.                      FOR SHAPE s IN This.InDataFlow s.InsertInFlow(GeneratePositionIn());
8.                      FOR SHAPE s IN This.OutDataFlow s.InsertOutFlow(GeneratePositionOut());
9.              }
10.             DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
11.             ThisWindow.IsChanged = FALSE;
```

```
}
```

DELETEDATAFLOW

```
ALGORITHM DeleteDataFlow(DATAFLOW df) : VOID {
    DELETE df FROM df.Start.Shape.OutDataFlow;
    DELETE df FROM df.End.Shape.InDataFlow;
    DELETE EDGE (df.Start.Shape, df.End.Shape) FROM ThisWindow.DFDBackend.UndirGraph;
    IF (df.Start.Shape.Type = DataProcess) {
        IF (df.Start.End.Type = DataProcess) {
            DELETE df FROM (
                ThisWindow.DFDBackend.Connects[
                    (df.Start.Shape, df.End.Shape)
                ]
            );
            DELETE df FROM (
                ThisWindow.DFDBackend.Connects[
                    (df.End.Shape, df.Start.Shape)
                ]
            );
        } ELSE {
            FOR (
                TWOPATH key
            ) in ThisWindow.DFDBackend.Connects.keys() {
                IF (
                    key.Uedge = df.End.Shape AND
                    (
                        ThisWindow.DFDBackend.Connects[
                            key
                        ].size > 0
                    )
                ) {
                    FOR (
                        PATH _
                    ) IN ThisWindow.DFDBackend.Connects[key] {
                        DELETE (
                            df, key
                        ) FROM (
                            ThisWindow.DFDBackend.Connects[
                                (
                                    df.Start.Shape,
                                    key.Vedge
                                )
                            ]
                        );
                    }
                }
            }
            DELETE df FROM (
                ThisWindow.DFDBackend.Connects[
                    (df.Start.Shape, df.End.Shape)
                ]
            );
```

```
50.                         }
51.                 } ELSE {
52.                         FOR TWOPATH key in ThisWindow.DFDBackend.Connects.keys() {
53.                                 IF (
54.                                         key.Vedge = s1 AND
55.                                         ThisWindow.DFDBackend.Connects[key].size > 0
56.                                 ) {
57.                                         FOR PATH _ IN ThisWindow.DFDBackend.Connects[key] {
58.                                                 DELETE (
59.                                                         NEW TWOPATH(key, df)
60.                                                 ) FROM (
61.                                                         ThisWindow.DFDBackend.Connects[
62.                                                                 (key.Uedge, s2)
63.                                                         ]
64.                                                 );
65.                                         }
66.                                 }
67.                         }
68.                         DELETE df FROM (
69.                                 ThisWindow.DFDBackend.Connects[
70.                                         (df.Start.Shape, df.End.Shape)
71.                                 ]
72.                         );
73.                 }
74.                 IF (df.Start.Decompose NOT NULL) ExtDeleteOutFlow(df);
75.                 IF (df.End.Decompose NOT NULL) ExtDeleteInFlow(df);
76.                 DELETE df.Name FROM ThisWindow.DDBackend.Names;
77.                 DELETE df FROM ThisWindow.DDBackend.Data;
78.                 DELETE df FROM ThisWindow.DFDBackend.Data;
79.                 DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
80.                 ThisWindow.IsChanged := TRUE;
81.         }
```

DELETEDATAPROCESS

```
1.      ALGORITHM DeleteDataProcess(DATAPROCESS dp) : VOID {
2.              FOR DATAFLOW df IN (dp.InDataFlow UNION dp.OutDataFlow) DeleteDataFlow(df);
3.              DELETE NODE dp FROM ThisWindow.DFDBackend.UndirGraph;
4.              FOR TWOPATH p IN ThisWindow.DFDBackend.Connects.keys() {
5.                      IF p IN dp DELETE p FROM ThisWindow.DFDBackend.Connects;
6.              }
7.              DELETE dp.Name FROM ThisWindow.DDBackend.Names;
8.              DELETE dp FROM ThisWindow.DDBackend.Data;
9.              DELETE dp FROM ThisWindow.DFDBackend.Data;
10.             DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
11.             ThisWindow.IsChanged := TRUE;
12.     }
```

DELETEDATASTORE

```
1.      ALGORITHM DeleteDataStore(DATASTORE ds) : VOID {
2.              FOR DATAFLOW df IN (ds.InDataFlow UNION ds.OutDataFlow) DeleteDataFlow(ds);
3.              DELETE NODE ds FROM ThisWindow.DFDBackend.UndirGraph;
4.              FOR TWOPATH tp IN ThisWindow.DFDBackend.Connects.keys() {
```

```
5.                          FOR PATH p IN ThisWindow.DFDBackend.Connects[ps] {
6.                              IF (p.Type = TWOPATH AND ds IN p) {
7.                                  DELETE p FROM ThisWindow.DFDBackend.Connects[tp];
8.                              }
9.                          }
10.                 }
11.                 DELETE ds.Name FROM ThisWindow.DDBackend.Names;
12.                 DELETE ds FROM ThisWindow.DDBackend.Data;
13.                 DELETE ds FROM ThisWindow.DFDBackend.Data;
14.                 DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
15.                 ThisWindow.IsChanged := TRUE;
16.     }
```

DELETEEXTERNALENTITY

```
1.      ALGORITHM DeleteExternalEntity(EXTERNALENTITY ee) : VOID {
2.              FOR DATAFLOW df IN (ee.InDataFlow UNION ee.OutDataFlow) DeleteDataFlow(ds);
3.              DELETE NODE ee FROM ThisWindow.DFDBackend.UndirGraph;
4.              FOR TWOPATH tp IN ThisWindow.DFDBackend.Connects.keys() {
5.                  FOR PATH p IN ThisWindow.DFDBackend.Connects[ps] {
6.                      IF (p.Type = TWOPATH AND ee IN p) {
7.                          DELETE p FROM ThisWindow.DFDBackend.Connects[tp];
8.                      }
9.                  }
10.             }
11.             DELETE ee.Name FROM ThisWindow.DDBackend.Names;
12.             DELETE ee FROM ThisWindow.DDBackend.Data;
13.             DELETE ee FROM ThisWindow.DFDBackend.Data;
14.             DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
15.             ThisWindow.IsChanged := TRUE;
16.     }
```

DELETEEXTERNALOUTPUT

```
1.      ALGORITHM DeleteExternalOutput(EXTERNALOUTPUT eo) : VOID {
2.              FOR DATAFLOW df IN eo.InDataFlow DeleteDataFlow(ds);
3.              DELETE NODE eo FROM ThisWindow.DFDBackend.UndirGraph;
4.              FOR TWOPATH tp IN ThisWindow.DFDBackend.Connects.keys() {
5.                  FOR PATH p IN ThisWindow.DFDBackend.Connects[ps] {
6.                      IF (p.Type = TWOPATH AND eo IN p) {
7.                          DELETE p FROM ThisWindow.DFDBackend.Connects[tp];
8.                      }
9.                  }
10.             }
11.             DELETE eo.Name FROM ThisWindow.DDBackend.Names;
12.             DELETE eo FROM ThisWindow.DDBackend.Data;
13.             DELETE eo FROM ThisWindow.DFDBackend.Data;
14.             DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
15.             ThisWindow.IsChanged := TRUE;
16.     }
```

DELETELABEL

```
1.      ALGORITHM DeleteLabel(LABEL l) : VOID {
2.              DELETE l FROM ThisWindow.DDBackend.Data;
```

```
3.              DELETE l FROM ThisWindow.DFDBackend.Data;
4.              DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
5.              ThisWindow.IsChanged := TRUE;
6.      }
```

EDITTEXT

```
1.      ALGORITHM EditText(LABEL l, STRING s) : VOID {
2.              l.Text := s;
3.              ThisWindow.IsChanged := TRUE;
4.      }
```

EXIT

```
1.      ALGORITHM Exit() : VOID {
2.              IF (NOT ThisWindow.IsChanged OR savePrompt()) {
3.                      pw := ThisWindow.Parent;
4.                      CLOSE ThisWindow;
5.                      ThisWindow := pw;
6.              }
7.      }
```

EXITDECOMPOSE

```
1.      ALGORITHM ExitDecompose() : VOID {
2.              IF (NOT ThisWindow.IsChanged OR savePromptDecompose()) {
3.                      pw := ThisWindow.Parent;
4.                      CLOSE ThisWindow;
5.                      ThisWindow := pw;
6.                      DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
7.                      ThisWindow.IsChanged := TRUE;
8.              }
9.      }
```

EXPORTDD

```
1.      ALGORITHM ExportDD() : VOID {
2.              dir, title := ChooseFile(“.txt”);
3.              OPEN dir/title AS file;
4.              file.DDBackend := ThisWindow.DDBackend;
5.              CLOSE file;
6.      }
```

EXPORTPNG

```
1.      ALGORITHM ExportPNG() : VOID {
2.              dir, title := ChooseFile(“.png”);
3.              OPEN dir/title AS file;
4.              DISPLAY ThisWindow.DFDBackend.Data IN file;
5.              CLOSE file;
6.      }
```

FINDSHAPEANCHOR

```
1.      ALGORITHM InsertDataFlow(POSITION P) : SHAPEANCHOR sa {
2.              listShapes := NEW ARRAY;
3.              FOR SHAPE s IN ThisWindow.DFDBackend.Data {
4.                      IF (s.Type != DataFlow AND s.Type != Label) {
```

```
5.                                  sP := s.Positions;
6.                                  IF (
7.                                          P[0] >= sP[0][0] AND
8.                                          P[0] <= sP[1][0] AND
9.                                          P[1] >= sP[0][1] AND
10.                                         P[1] <= sP[1][1]
11.                                 ) {
12.                                         SWITCH(
13.                                                 minIdx(
14.                                                         {
15.                                                                 P[0] – sP[0][0],
16.                                                                 sP[1][0] – P[0],
17.                                                                 P[1] – sP[0][1],
18.                                                                 sP[1][1] – P[1]
19.                                                         }
20.                                                 )
21.                                         ) {
22.                                                 CASE 0: ADD (s, Left) IN listShapes;
23.                                                 CASE 1: ADD (s, Right) IN listShapes;
24.                                                 CASE 2: ADD (s, Up) IN listShapes;
25.                                                 CASE 3: ADD (s, Down) IN listShapes;
26.                                         }
27.                                 }
28.                         }
29.                 }
30.                 IF (listShapes.Size == 0) sa := NULL;
31.                 ELSE IF (listShapes.Size == 1) sa := listShapes[0];
32.                 ELSE sa := chooseShape(listShapes);
33.         }
```

HELP

```
1.      ALGORITHM HELP() : VOID {OPEN Help dialog box;}
```

INSERTDATAFLOW

```
1.      ALGORITHM InsertDataFlow(POSITIONS P) : VOID {
2.              df := NEW DATAFLOW;
3.              IF (ValPositions(P)) {
4.                      df.Start := FindShapeAnchor(P[0]);
5.                      df.End := FindShapeAnchor(P[1]);
6.                      IF ((df.Start NOT NULL) AND (df.End NOT NULL)) {
7.                              IF (
8.                                      (
9.                                              df.Start.Shape.Type = DataProcess OR
10.                                             df.End.Shape.Type = DataProcess
11.                                     ) AND df.Start.Shape.Type != ExternalOutput
12.                             ) {
13.                                     updateConnects(df);
14.                                     IF (df.Start.Shape.Type = DataProcess) {
15.                                             IF (df.End.Shape.Type = DataProcess) df.DDWizard(0);
16.                                             ELSE df.DDWizard(1);
17.                                     } ELSE df.DDWizard(2);
18.                                     df.Decompose := NULL;
```

```
                                        ADD df IN df.Start.Shape.OutDataFlow;
                                        ADD df IN df.End.Shape.InDataFlow;
                                        ADD df IN ThisWindow.DFDBackend.Data;
                                        ADD (
                                                EDGE (df.Start.Shape, df.End.Shape)
                                        ) IN ThisWindow.DFDBackend.UndirGraph;
                                        DISPLAY (
                                                ThisWindow.DFDBackend.Data
                                        ) IN ThisWindow.Workspace;
                                        ThisWindow.IsChanged = TRUE;
                                }
                        }
                }
        }
```

INSERTDATAPROCESS

```
ALGORITHM InsertDataProcess(POSITIONS P) : VOID {
        dp := NEW DATAPROCESS;
        IF (ValPositions(P)) {
                INPUT dp.Name;
                IF (dp.Name IN ThisWindow.DDBackend.Names) {
                        errorNameExists();
                        GOTO 4;
                } ELSE ADD dp.Name IN ThisWindow.DDBackend.Names;
                dp.Positions := P;
                dp.Decompose := NULL;
                ADD dp IN ThisWindow.DFDBackend.Data;
                ADD NODE dp IN ThisWindow.DFDBackend.UndirGraph;
                DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
                ThisWindow.IsChanged = TRUE;
        }
}
```

INSERTDATASTORE

```
ALGORITHM InsertDataStore(POSITIONS P) : VOID {
        ds := NEW DATASTORE;
        IF (ValPositions(P)) {
                INPUT ds.Name;
                IF (ds.Name IN ThisWindow.DDBackend.Names) {
                        errorNameExists();
                        GOTO 4;
                } ELSE ADD ds.Name IN ThisWindow.DDBackend.Names;
                ds.Positions := P;
                ds.Decompose := NULL;
                ADD ds IN ThisWindow.DFDBackend.Data;
                ADD NODE ds IN ThisWindow.DFDBackend.UndirGraph;
                DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
                ThisWindow.IsChanged = TRUE;
        }
}
```

INSERTEXTERNALENTITY

```
1.  ALGORITHM InsertExternalEntity(POSITIONS P) : VOID {
2.      ee := NEW EXTERNALENTITY;
3.      IF (ValPositions(P)) {
4.          INPUT ee.Name;
5.          IF (ee.Name IN ThisWindow.DDBackend.Names) {
6.              errorNameExists();
7.              GOTO 4;
8.          } ELSE ADD ee.Name IN ThisWindow.DDBackend.Names;
9.          ee.Positions := P;
10.         ee.Decompose := NULL;
11.         ADD ee IN ThisWindow.DFDBackend.Data;
12.         ADD NODE ee IN ThisWindow.DFDBackend.UndirGraph;
13.         DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
14.         ThisWindow.IsChanged = TRUE;
15.     }
16. }
```

INSERTEXTERNALOUTPUT

```
1.  ALGORITHM InsertExternalOutput(POSITIONS P) : VOID {
2.      eo := NEW EXTERNALOUTPUT;
3.      IF (ValPositions(P)) {
4.          INPUT eo.Name;
5.          IF (eo.Name IN ThisWindow.DDBackend.Names) {
6.              errorNameExists();
7.              GOTO 4;
8.          } ELSE ADD eo.Name IN ThisWindow.DDBackend.Names;
9.          eo.Positions := P;
10.         eo.Decompose := NULL;
11.         ADD eo IN ThisWindow.DFDBackend.Data;
12.         ADD NODE eo IN ThisWindow.DFDBackend.UndirGraph;
13.         DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
14.         ThisWindow.IsChanged = TRUE;
15.     }
16. }
```

INSERTINFLOW

```
1.  ALGORITHM InsertInFlow(POSITIONS P) : VOID {
2.      if := NEW INFLOW;
3.      IF (ValPositions(P)) {
4.          if.Name := This.Name;
5.          if.Positions := P;
6.          if.Decompose := NULL;
7.          ADD if IN ThisWindow.DFDBackend.Data;
8.          ADD NODE if IN ThisWindow.DFDBackend.UndirGraph;
9.          DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
10.     }
11. }
```

INSERTOUTFLOW

```
1.  ALGORITHM InsertOutFlow(POSITIONS P) : VOID {
2.      of := NEW OUTFLOW;
3.      IF (ValPositions(P)) {
```

```
4.                          of.Name := This.Name;
5.                          of.Positions := P;
6.                          if.Decompose := NULL;
7.                          ADD of IN ThisWindow.DFDBackend.Data;
8.                          ADD NODE of IN ThisWindow.DFDBackend.UndirGraph;
9.                          DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
10.                 }
11.         }
```

INSERTLABEL

```
1.          ALGORITHM InsertLabel(POSITIONS P) : VOID {
2.                  l := NEW LABEL;
3.                  IF (ValPositions(P)) {
4.                          INPUT l.Text;
5.                          l.Positions := P;
6.                          l.Decompose := NULL;
7.                          ADD l IN ThisWindow.DFDBackend.Data;
8.                          DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace;
9.                          ThisWindow.IsChanged = TRUE;
10.                 }
11.         }
```

MOVESHAPE

```
1.          ALGORITHM MoveShape(SHAPE S, POSITIONS P) : VOID {
2.                  IF (ValPositions(P)) {
3.                          ThisWindow.DFDBackend.Data.S.Positions := P;
4.                          DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.WorkSpace;
5.                          ThisWindow.IsChanged = TRUE;
6.                  }
7.          }
```

NEWFILE

```
1       ALGORITHM NewFile() : VOID {
2               IF (NOT ThisWindow.IsChanged OR savePrompt()) {
3                       CLEAR ThisWindow.Workspace;
4                       ThisWindow.Title := “Untitled”;
5                       CLEAR ThisWindow.DFDBackend;
6                       CLEAR ThisWindow.DDBackend;
7                       CLEAR ThisWindow.Log;
8               }
9       }
```

OPENFILE

```
1.          ALGORITHM OpenFile() : VOID {
2.                  IF (NOT ThisWindow.IsChanged OR savePrompt()) {
3.                          ThisWindow.Dir, ThisWindow.Title := ChooseFile(“.dfd”);
4.                          OPEN ThisWindow.Dir/ThisWindow.Title AS file;
5.                          DISPLAY file.DFDBackend.Data IN ThisWindow.Workspace;
6.                          ThisWindow.DFDBackend := file.DFDBackend;
7.                          ThisWindow.DDBackend := file.DDBackend;
8.                          CLEAR ThisWindow.Log;
9.                          CLOSE file;
```

```
10.             }
11.     }
```

PRINTFILE

```
1.      ALGORITHM PrintFile() : VOID {
2.              IF (NOT ThisWindow.IsChanged OR savePrompt()) {
3.                      OPEN "~.png" AS tmp;
4.                      DISPLAY ThisWindow.DFDBackend.Data IN tmp;
5.                      PRINT tmp;
6.                      CLOSE tmp;
7.                      DELETE "~.png";
8.              }
9.      }
```

RENAMEDATAFLOW

```
1.      ALGORITHM RenameDataFlow(DATAFLOW df, STRING NewName) : VOID {
2.              IF (df.Start.Decompose NOT NULL) {
3.                      RenameShape(
4.                              df.Start.Decompose.DFDBackend.This.OutDataFlow.df, NewName
5.                      );
6.              }
7.              IF (df.End.Decompose NOT NULL) {
8.                      RenameShape(
9.                              df.End.Decompose.DFDBackend.This.InDataFlow.df, NewName
10.                     );
11.             }
12.             RenameShape(If, NewName);
13.             ThisWindow.IsChanged = TRUE;
14.     }
```

RENAMEINFLOW

```
1.      ALGORITHM RenameInFlow(INFLOW If, STRING NewName) : VOID {
2.              RenameShape(This.InDataFlow.If, NewName);
3.              RenameShape(If, NewName);
4.              ThisWindow.ParentWindow.IsChanged = TRUE;
5.      }
```

RENAMEOUTFLOW

```
1.      ALGORITHM RenameOutFlow(OUTFLOW Of, STRING NewName) : VOID {
2.              RenameShape(This.OutDataFlow.Of, NewName);
3.              RenameShape(Of, NewName);
4.              ThisWindow.ParentWindow.IsChanged = TRUE;
5.      }
```

RENAMESHAPE

```
1.      ALGORITHM RenameShape(SHAPE S, STRING NewName) : VOID {
2.              ThisWindow.DFDBackend.Data.S.Name := NewName;
3.              DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.WorkSpace;
4.              ThisWindow.IsChanged = TRUE;
5.      }
```

RESIZESHAPE

```
1.  ALGORITHM ResizeShape(SHAPE S, POSITIONS P) : VOID {
2.      IF (ValPositions(P)) {
3.          ThisWindow.DFDBackend.Data.S.Positions := P;
4.          DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.WorkSpace;
5.          ThisWindow.IsChanged = TRUE;
6.      }
7.  }
```

SaveAs

```
1.  ALGORITHM SaveAs() : BOOLEAN Ans {
2.      IF (NOT DebugFile()) {
3.          dir, title := ChooseFile(".dfd");
4.          OPEN dir/title AS file;
5.          file.DFDBackend := ThisWindow.DFDBackend;
6.          file.DDBackend := ThisWindow.DDBackend;
7.          ThisWindow.IsChanged := TRUE;
8.          CLOSE file;
9.          Ans := TRUE;
10.     } ELSE Ans := FALSE;
11. }
```

SaveDecompose

```
1.  ALGORITHM SaveDecompose() : BOOLEAN Ans {
2.      IF (NOT DebugFile()) {
3.          This.DFDBackend := This.DFDBackend;
4.          This.DDBackend := ThisWindow.DDBackend;
5.          ThisWindow.IsChanged := TRUE;
6.      } ELSE Ans := FALSE;
7.  }
```

SaveFile

```
1.  ALGORITHM SaveFile() : BOOLEAN Ans {
2.      IF (NOT DebugFile()) {
3.          IF (ThisWindow.Title = "Untitled") SaveAs(); ELSE {
4.              OPEN ThisWindow.Dir/ThisWindow.Title AS file;
5.              file.DFDBackend := ThisWindow.DFDBackend;
6.              file.DDBackend := ThisWindow.DDBackend;
7.              ThisWindow.IsChanged := TRUE;
8.              CLOSE file;
9.          }
10.         Ans := TRUE;
11.     } ELSE Ans := FALSE;
12. }
```

SavePrompt

```
1.  ALGORITHM SavePrompt() : BOOLEAN Ans {
2.      OPEN SavePrompt dialog box AS sp;
3.      Ans := ((sp.Output = "Yes" AND SaveFile()) OR sp.Output = "No");
4.  }
```

SavePromptDecompose

```
1.  ALGORITHM SavePromptDecompose() : BOOLEAN Ans {
```

```
2.              OPEN SavePrompt dialog box AS sp;
3.              Ans := ((sp.Output = “Yes” AND SaveDecompose()) OR sp.Output = “No”);
4.      }
```

UPDATECONNECTS

```
1.      ALGORITHM UpdateConnects(DATAFLOW df) : VOID {
2.              s1 := df.Start.Shape;
3.              s2 := df.End.Shape;
4.              IF (s1.Type = DataProcess) {
5.                      IF (s2.Type = DataProcess) {
6.                              ADD df IN ThisWindow.DFDBackend.Connects[(s1, s2)];
7.                              ADD df IN ThisWindow.DFDBackend.Connects[(s2, s1)];
8.                      } ELSE {
9.                              ADD df IN ThisWindow.DFDBackend.Connects[(s1, s2)];
10.                             FOR TWOPATH key in ThisWindow.DFDBackend.Connects.keys() {
11.                                     IF (
12.                                             key.Uedge = s2 AND
13.                                             ThisWindow.DFDBackend.Connects[key].size > 0
14.                                     ) {
15.                                             FOR (
16.                                                     PATH _
17.                                             ) IN ThisWindow.DFDBackend.Connects[key] {
18.                                                     ADD (
19.                                                             NEW TWOPATH(df, key)
20.                                                     ) IN (
21.                                                             ThisWindow.DFDBackend.Connects[
22.                                                                     (s1, key.Vedge)
23.                                                             ]
24.                                                     );
25.                                             }
26.                                     }
27.                             }
28.                     }
29.             } ELSE {
30.                     ADD df IN ThisWindow.DFDBackend.Connects[(s1, s2)];
31.                     FOR TWOPATH key in ThisWindow.DFDBackend.Connects.keys() {
32.                             IF (
33.                                     key.Vedge = s1 AND
34.                                     ThisWindow.DFDBackend.Connects[key].size > 0
35.                             ) {
36.                                     FOR PATH _ IN ThisWindow.DFDBackend.Connects[key] {
37.                                             ADD (
38.                                                     NEW TWOPATH(key, df)
39.                                             ) IN (
40.                                                     ThisWindow.DFDBackend.Connects[
41.                                                             (key.Uedge, s2)
42.                                                     ]
43.                                             );
44.                                     }
45.                             }
46.                     }
47.             }
48.     }
```

VALPOSITIONS

```
1.	ALGORITHM ValPositions(POSITIONS P) : BOOLEAN Ans {
2.		wp := ThisWindow.Workspace.Positions;
3.		ans := ALL(
4.			(
5.				(p[0] >= wp[0][0] AND p[0] <= wp[1][0]) AND
6.				(p[1] >= wp[0][1] AND p[1] <= wp[1][1])
7.			) FOR POSITION p IN P
8.		);
9.	}
```