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

|  |  |
| --- | --- |
|  | ALGORITHM DDWizard(INTEGER checkCases) : VOID { |
|  | OPEN DDWizard dialog box AS dw; |
|  | okDisabled := TRUE; |
|  | ROLLBACK IF (dw.cancelPressed()) { |
|  | INPUT dw.Name; |
|  | IF (dw.Name IN ThisWindow.DDBackend.Names) { |
|  | IF (checkCases = 0) { |
|  | errorNameExists(); |
|  | GOTO 5; |
|  | } ELSE IF (checkCases = 1) { |
|  | n := This.End.Names; |
|  | IF (dw.Name IN n) { |
|  | This.Name := dw.Name; |
|  | This.Type := This.End.Names[dw.Name].Type; |
|  | } ELSE { |
|  | errorNameExists(); |
|  | GOTO 5; |
|  | } |
|  | } ELSE { |
|  | n := This.Start.Names; |
|  | IF (dw.Name IN n) { |
|  | This.Name := dw.Name; |
|  | This.Type := This.Start.Names[dw.Name].Type; |
|  | } ELSE { |
|  | errorNameExists(); |
|  | GOTO 5; |
|  | } |
|  | } |
|  | } ELSE { |
|  | INPUT dw.Type; |
|  | IF (dw.Type = Struct) CreateStruct(dw.Name); |
|  | This.Name := dw.Name; |
|  | This.Type := dw.Type; |
|  | ADD dw.Name IN ThisWindow.DDBackend.Names; |
|  | IF (checkCases = 1) ADD dw.Name IN This.End.Names; |
|  | ELSE IF (checkCases = 2) ADD dw.Name IN This.Start.Names; |
|  | } |
|  | okDisabled := FALSE; |
|  | WHILE (NOT dw.okPressed()); |
|  | } |
|  | CLOSE dw; |
|  | } |

## DebugFile

|  |  |
| --- | --- |
|  | ALGORITHM DebugFile() : BOOLEAN Ans { |
|  | du := DebugUnnecessary(); |
|  | di := DebugIsolated(); |
|  | db := DebugBalanced(); |
|  | Ans := du OR di OR db; |
|  | } |

## DebugBalanced

|  |  |
| --- | --- |
|  | ALGORITHM DebugBalanced() : BOOLEAN Ans { |
|  | Ans := FALSE; |
|  | FOR DATAPROCESS dp IN ThisWindow.DFDBackend.Data { |
|  | Ans := Ans OR s.DebugBalancedShape(); |
|  | } |
|  | } |

## DebugBalancedShape

|  |  |
| --- | --- |
|  | ALGORITHM DebugBalancedShape() : BOOLEAN Ans { |
|  | Ans := FALSE; |
|  | IF (This.Decompose NOT NULL) { |
|  | IF ( |
|  | ( |
|  | SET( |
|  | FOR SHAPE s2 IN This.InDataFlow |
|  | ) != SET( |
|  | FOR INFLOW if IN This.Decompose.DFDBackend.Data |
|  | ) |
|  | ) OR |
|  | ( |
|  | SET( |
|  | FOR SHAPE s2 IN This.OutDataFlow |
|  | ) != SET( |
|  | FOR ( |
|  | OUTFLOW of |
|  | ) IN This.Decompose.DFDBackend.Data |
|  | ) |
|  | ) |
|  | ) { |
|  | PRINT “Balancing Error in “ + This.Name + “\n” IN ThisWindow.Log; |
|  | Ans := TRUE; |
|  | } |
|  | } |
|  | } |

## DebugIsolated

|  |  |
| --- | --- |
|  | ALGORITHM DebugIsolated() : BOOLEAN Ans { |
|  | g := ThisWindow.DFDBackend.UndirGraph; |
|  | n := g.Nodes.length; |
|  | IF (n == 0) Ans := FALSE; ELSE { |
|  | isVisited := {s: FALSE FOR SHAPE s IN g}; |
|  | s0 := RANDOMCHOICE(g.Nodes); |
|  | isVisited[s0] := TRUE; |
|  | QUEUE q; |
|  | q.Enqueue(0); |
|  | REPEAT UNTIL (q.IsEmpty()) { |
|  | u := q.Dequeue(); |
|  | isVisited[u] := TRUE; |
|  | FOR EDGE e FROM u {IF (NOT isVisited[e.v]) q.Enqueue(e.v);) |
|  | } |
|  | FOR SHAPE s, BOOLEAN b IN ENUMERATE(isVisited) { |
|  | IF NOT b { |
|  | PRINT ( |
|  | “Isolation Error in “ + s.Name + “\n” |
|  | ) IN ThisWindow.Log; |
|  | Ans := TRUE; |
|  | } |
|  | } |
|  | } |
|  | } |

## DebugUnnecessary

|  |  |
| --- | --- |
|  | ALGORITHM DebugUnnecessary() : BOOLEAN Ans { |
|  | mc := ThisWindow.DFDBackend.Connects; |
|  | FOR TWOPATH x IN mc.keys() { |
|  | IF (mc[x].length() > 1) { |
|  | PRINT (“Unnecessary flow in“ + mc + “\n”) IN ThisWindow.Log; |
|  | Ans := TRUE; |
|  | } |
|  | } |
|  | } |

## Decompose

|  |  |
| --- | --- |
|  | ALGORITHM Decompose() : VOID { |
|  | OPEN NEW DECOMPOSEWINDOW as dw; |
|  | dw.Parent := ThisWindow; |
|  | ThisWindow := dw; |
|  | ThisWindow.Title := This.Name + “\_Decompose”; |
|  | IF (This.Decompose NULL) { |
|  | FOR SHAPE s IN This.InDataFlow s.InsertInFlow(GeneratePositionIn()); |
|  | FOR SHAPE s IN This.OutDataFlow s.InsertOutFlow(GeneratePositionOut()); |
|  | } |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace; |
|  | 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) |
|  | ] |
|  | ); |
|  | } |
|  | } ELSE { |
|  | FOR TWOPATH key in ThisWindow.DFDBackend.Connects.keys() { |
|  | IF ( |
|  | key.Vedge = s1 AND |
|  | ThisWindow.DFDBackend.Connects[key].size > 0 |
|  | ) { |
|  | FOR PATH \_ IN ThisWindow.DFDBackend.Connects[key] { |
|  | DELETE ( |
|  | NEW TWOPATH(key, df) |
|  | ) FROM ( |
|  | ThisWindow.DFDBackend.Connects[ |
|  | (key.Uedge, s2) |
|  | ] |
|  | ); |
|  | } |
|  | } |
|  | } |
|  | DELETE df FROM ( |
|  | ThisWindow.DFDBackend.Connects[ |
|  | (df.Start.Shape, df.End.Shape) |
|  | ] |
|  | ); |
|  | } |
|  | IF (df.Start.Decompose NOT NULL) ExtDeleteOutFlow(df); |
|  | IF (df.End.Decompose NOT NULL) ExtDeleteInFlow(df); |
|  | DELETE df.Name FROM ThisWindow.DDBackend.Names; |
|  | DELETE df FROM ThisWindow.DDBackend.Data; |
|  | DELETE df FROM ThisWindow.DFDBackend.Data; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace; |
|  | ThisWindow.IsChanged := TRUE; |
|  | } |

## DeleteDataProcess

|  |  |
| --- | --- |
|  | ALGORITHM DeleteDataProcess(DATAPROCESS dp) : VOID { |
|  | FOR DATAFLOW df IN (dp.InDataFlow UNION dp.OutDataFlow) DeleteDataFlow(df); |
|  | DELETE NODE dp FROM ThisWindow.DFDBackend.UndirGraph; |
|  | FOR PATH p IN ThisWindow.DFDBackend.Connects.keys() { |
|  | IF (p.Type = ONEPATH AND p = dp OR p.Type = TWOPATH AND p IN dp) { |
|  | DELETE p FROM ThisWindow.DFDBackend.Connects; |
|  | } |
|  | } |
|  | DELETE dp.Name FROM ThisWindow.DDBackend.Names; |
|  | DELETE dp FROM ThisWindow.DDBackend.Data; |
|  | DELETE dp FROM ThisWindow.DFDBackend.Data; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace; |
|  | ThisWindow.IsChanged := TRUE; |
|  | } |

## DeleteDataStore

|  |  |
| --- | --- |
|  | ALGORITHM DeleteDataStore(DATASTORE ds) : VOID { |
|  | FOR DATAFLOW df IN (ds.InDataFlow UNION ds.OutDataFlow) DeleteDataFlow(ds); |
|  | DELETE NODE ds FROM ThisWindow.DFDBackend.UndirGraph; |
|  | FOR PATH p IN ThisWindow.DFDBackend.Connects.keys() { |
|  | IF (p.Type = ONEPATH AND p = ds OR p.Type = TWOPATH AND p IN ds) { |
|  | DELETE p FROM ThisWindow.DFDBackend.Connects; |
|  | } |
|  | } |
|  | DELETE ds.Name FROM ThisWindow.DDBackend.Names; |
|  | DELETE ds FROM ThisWindow.DDBackend.Data; |
|  | DELETE ds FROM ThisWindow.DFDBackend.Data; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace; |
|  | ThisWindow.IsChanged := TRUE; |
|  | } |

## DeleteExternalEntity

|  |  |
| --- | --- |
|  | ALGORITHM DeleteExternalEntity(EXTERNALENTITY ee) : VOID { |
|  | FOR DATAFLOW df IN (ee.InDataFlow UNION ee.OutDataFlow) DeleteDataFlow(ds); |
|  | DELETE NODE ee FROM ThisWindow.DFDBackend.UndirGraph; |
|  | FOR PATH p IN ThisWindow.DFDBackend.Connects.keys() { |
|  | IF (p.Type = ONEPATH AND p = ee OR p.Type = TWOPATH AND p IN ee) { |
|  | DELETE p FROM ThisWindow.DFDBackend.Connects; |
|  | } |
|  | } |
|  | DELETE ee.Name FROM ThisWindow.DDBackend.Names; |
|  | DELETE ee FROM ThisWindow.DDBackend.Data; |
|  | DELETE ee FROM ThisWindow.DFDBackend.Data; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace; |
|  | ThisWindow.IsChanged := TRUE; |
|  | } |

## DeleteLabel

|  |  |
| --- | --- |
|  | ALGORITHM DeleteLabel(LABEL l) : VOID { |
|  | DELETE l FROM ThisWindow.DDBackend.Data; |
|  | DELETE l FROM ThisWindow.DFDBackend.Data; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace; |
|  | ThisWindow.IsChanged := TRUE; |
|  | } |

## EditText

|  |  |
| --- | --- |
|  | ALGORITHM EditText(LABEL l, STRING s) : VOID { |
|  | l.Text := s; |
|  | ThisWindow.IsChanged := TRUE; |
|  | } |

## Exit

|  |  |
| --- | --- |
|  | ALGORITHM Exit() : VOID { |
|  | IF (NOT ThisWindow.IsChanged OR savePrompt()) { |
|  | pw := ThisWindow.Parent; |
|  | CLOSE ThisWindow; |
|  | ThisWindow := pw; |
|  | } |
|  | } |

## ExitDecompose

|  |  |
| --- | --- |
|  | ALGORITHM ExitDecompose() : VOID { |
|  | IF (NOT ThisWindow.IsChanged OR savePromptDecompose()) { |
|  | pw := ThisWindow.Parent; |
|  | CLOSE ThisWindow; |
|  | ThisWindow := pw; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace; |
|  | ThisWindow.IsChanged := TRUE; |
|  | } |
|  | } |

## ExportDD

|  |  |
| --- | --- |
|  | ALGORITHM ExportDD() : VOID { |
|  | dir, title := ChooseFile(“.txt”); |
|  | OPEN dir/title AS file; |
|  | file.DDBackend := ThisWindow.DDBackend; |
|  | CLOSE file; |
|  | } |

## ExportPNG

|  |  |
| --- | --- |
|  | ALGORITHM ExportPNG() : VOID { |
|  | dir, title := ChooseFile(“.png”); |
|  | OPEN dir/title AS file; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN file; |
|  | CLOSE file; |
|  | } |

## FindShapeAnchor

|  |  |
| --- | --- |
|  | ALGORITHM InsertDataFlow(POSITION P) : SHAPEANCHOR sa { |
|  | listShapes := NEW ARRAY; |
|  | FOR SHAPE s IN ThisWindow.DFDBackend.Data { |
|  | IF (s.Type != DataFlow AND s.Type != Label) { |
|  | sP := s.Positions; |
|  | IF ( |
|  | P[0] >= sP[0][0] AND |
|  | P[0] <= sP[1][0] AND |
|  | P[1] >= sP[0][1] AND |
|  | P[1] <= sP[1][1] |
|  | ) { |
|  | SWITCH( |
|  | minIdx( |
|  | { |
|  | P[0] – sP[0][0], |
|  | sP[1][0] – P[0], |
|  | P[1] – sP[0][1], |
|  | sP[1][1] – P[1] |
|  | } |
|  | ) |
|  | ) { |
|  | CASE 0: ADD (s, Left) IN listShapes; |
|  | CASE 1: ADD (s, Right) IN listShapes; |
|  | CASE 2: ADD (s, Up) IN listShapes; |
|  | CASE 3: ADD (s, Down) IN listShapes; |
|  | } |
|  | } |
|  | } |
|  | } |
|  | IF (listShapes.Size == 0) sa := NULL; |
|  | ELSE IF (listShapes.Size == 1) sa := listShapes[0]; |
|  | ELSE sa := chooseShape(listShapes); |
|  | } |

## Help

|  |  |
| --- | --- |
|  | ALGORITHM HELP() : VOID {OPEN Help dialog box;} |

## InsertDataFlow

|  |  |
| --- | --- |
|  | ALGORITHM InsertDataFlow(POSITIONS P) : VOID { |
|  | df := NEW DATAFLOW; |
|  | IF (ValPositions(P)) { |
|  | df.Start := FindShapeAnchor(P[0]); |
|  | df.End := FindShapeAnchor(P[1]); |
|  | IF ((df.Start NOT NULL) AND (df.End NOT NULL)) { |
|  | IF ( |
|  | df.Start.Shape.Type = DataProcess OR |
|  | df.End.Shape.Type = DataProcess |
|  | ) { |
|  | updateConnects(df); |
|  | IF (df.Start.Shape.Type = DataProcess) { |
|  | IF (df.End.Shape.Type = DataProcess) df.DDWizard(0); |
|  | ELSE df.DDWizard(1); |
|  | } ELSE df.DDWizard(2); |
|  | 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

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

## InsertExternalOutput

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

## InsertInFlow

|  |  |
| --- | --- |
|  | ALGORITHM InsertInFlow(POSITIONS P) : VOID { |
|  | if := NEW INFLOW; |
|  | IF (ValPositions(P)) { |
|  | if.Name := This.Name; |
|  | if.Positions := P; |
|  | if.Decompose := NULL; |
|  | ADD if IN ThisWindow.DFDBackend.Data; |
|  | ADD NODE if IN ThisWindow.DFDBackend.UndirGraph; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace; |
|  | } |
|  | } |

## InsertOutFlow

|  |  |
| --- | --- |
|  | ALGORITHM InsertOutFlow(POSITIONS P) : VOID { |
|  | of := NEW OUTFLOW; |
|  | IF (ValPositions(P)) { |
|  | of.Name := This.Name; |
|  | of.Positions := P; |
|  | if.Decompose := NULL; |
|  | ADD of IN ThisWindow.DFDBackend.Data; |
|  | ADD NODE of IN ThisWindow.DFDBackend.UndirGraph; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace; |
|  | } |
|  | } |

## InsertLabel

|  |  |
| --- | --- |
|  | ALGORITHM InsertLabel(POSITIONS P) : VOID { |
|  | l := NEW LABEL; |
|  | IF (ValPositions(P)) { |
|  | INPUT l.Text; |
|  | l.Positions := P; |
|  | l.Decompose := NULL; |
|  | ADD l IN ThisWindow.DFDBackend.Data; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.Workspace; |
|  | ThisWindow.IsChanged = TRUE; |
|  | } |
|  | } |

## MoveShape

|  |  |
| --- | --- |
|  | ALGORITHM MoveShape(SHAPE S, POSITIONS P) : VOID { |
|  | IF (ValPositions(P)) { |
|  | ThisWindow.DFDBackend.Data.S.Positions := P; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.WorkSpace; |
|  | ThisWindow.IsChanged = TRUE; |
|  | } |
|  | } |

## 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

|  |  |
| --- | --- |
|  | ALGORITHM OpenFile() : VOID { |
|  | IF (NOT ThisWindow.IsChanged OR savePrompt()) { |
|  | ThisWindow.Dir, ThisWindow.Title := ChooseFile(“.dfd”); |
|  | OPEN ThisWindow.Dir/ThisWindow.Title AS file; |
|  | DISPLAY file.DFDBackend.Data IN ThisWindow.Workspace; |
|  | ThisWindow.DFDBackend := file.DFDBackend; |
|  | ThisWindow.DDBackend := file.DDBackend; |
|  | CLEAR ThisWindow.Log; |
|  | CLOSE file; |
|  | } |
|  | } |

## PrintFile

|  |  |
| --- | --- |
|  | ALGORITHM PrintFile() : VOID { |
|  | IF (NOT ThisWindow.IsChanged OR savePrompt()) { |
|  | OPEN “~.png” AS tmp; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN tmp; |
|  | PRINT tmp; |
|  | CLOSE tmp; |
|  | DELETE “~.png”; |
|  | } |
|  | } |

## RenameDataFlow

|  |  |
| --- | --- |
|  | ALGORITHM RenameDataFlow(DATAFLOW df, STRING NewName) : VOID { |
|  | IF (df.Start.Decompose NOT NULL) { |
|  | RenameShape( |
|  | df.Start.Decompose.DFDBackend.This.OutDataFlow.df, NewName |
|  | ); |
|  | } |
|  | IF (df.End.Decompose NOT NULL) { |
|  | RenameShape( |
|  | df.End.Decompose.DFDBackend.This.InDataFlow.df, NewName |
|  | ); |
|  | } |
|  | RenameShape(If, NewName); |
|  | ThisWindow.IsChanged = TRUE; |
|  | } |

## RenameInFlow

|  |  |
| --- | --- |
|  | ALGORITHM RenameInFlow(INFLOW If, STRING NewName) : VOID { |
|  | RenameShape(This.InDataFlow.If, NewName); |
|  | RenameShape(If, NewName); |
|  | ThisWindow.ParentWindow.IsChanged = TRUE; |
|  | } |

## RenameOutFlow

|  |  |
| --- | --- |
|  | ALGORITHM RenameOutFlow(OUTFLOW Of, STRING NewName) : VOID { |
|  | RenameShape(This.OutDataFlow.Of, NewName); |
|  | RenameShape(Of, NewName); |
|  | ThisWindow.ParentWindow.IsChanged = TRUE; |
|  | } |

## RenameShape

|  |  |
| --- | --- |
|  | ALGORITHM RenameShape(SHAPE S, STRING NewName) : VOID { |
|  | ThisWindow.DFDBackend.Data.S.Name := NewName; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.WorkSpace; |
|  | ThisWindow.IsChanged = TRUE; |
|  | } |

## ResizeShape

|  |  |
| --- | --- |
|  | ALGORITHM ResizeShape(SHAPE S, POSITIONS P) : VOID { |
|  | IF (ValPositions(P)) { |
|  | ThisWindow.DFDBackend.Data.S.Positions := P; |
|  | DISPLAY ThisWindow.DFDBackend.Data IN ThisWindow.WorkSpace; |
|  | ThisWindow.IsChanged = TRUE; |
|  | } |
|  | } |

## RotateShape

|  |  |
| --- | --- |
|  | ALGORITHM RotateShape(SHAPE S) : BOOLEAN Ans { |
|  | p := S.Positions; |
|  | rx1 := (p[0][0] + p[1][0]) / 2; |
|  | rx2 := (p[0][0] – p[1][0]) / 2; |
|  | ry1 := (p[0][1] + p[1][1]) / 2; |
|  | ry2 := (p[0][1] – p[1][1]) / 2; |
|  | x1 := rx1 + ry2; |
|  | y1 := rx2 + ry1; |
|  | x2 := rx1 – ry2; |
|  | y2 := rx2 – ry1; |
|  | pNew := ((x1, y1), (x2, y2)); |
|  | IF (ValPositions(pNew)) { |
|  | d := ThisWindow.DFDBackend.Data; |
|  | d.S.Positions := pNew; |
|  | d.S.Rotate = (d.S.Rotate + 1) mod 4; |
|  | DISPLAY d IN ThisWindow.WorkSpace; |
|  | ThisWindow.IsChanged = TRUE; |
|  | Ans := TRUE; |
|  | } ELSE { |
|  | ERROR(“Rotated values not valid”); |
|  | Ans := FALSE; |
|  | } |
|  | } |

## SaveAs

|  |  |
| --- | --- |
|  | ALGORITHM SaveAs() : BOOLEAN Ans { |
|  | IF (NOT DebugFile()) { |
|  | dir, title := ChooseFile(“.dfd”); |
|  | OPEN dir/title AS file; |
|  | file.DFDBackend := ThisWindow.DFDBackend; |
|  | file.DDBackend := ThisWindow.DDBackend; |
|  | ThisWindow.IsChanged := TRUE; |
|  | CLOSE file; |
|  | Ans := TRUE; |
|  | } ELSE Ans := FALSE; |
|  | } |

## SaveDecompose

|  |  |
| --- | --- |
|  | ALGORITHM SaveDecompose() : BOOLEAN Ans { |
|  | IF (NOT DebugFile()) { |
|  | This.DFDBackend := This.DFDBackend; |
|  | This.DDBackend := ThisWindow.DDBackend; |
|  | ThisWindow.IsChanged := TRUE; |
|  | } ELSE Ans := FALSE; |
|  | } |

## SaveFile

|  |  |
| --- | --- |
|  | ALGORITHM SaveFile() : BOOLEAN Ans { |
|  | IF (NOT DebugFile()) { |
|  | IF (ThisWindow.Title = “Untitled”) SaveAs(); ELSE { |
|  | OPEN ThisWindow.Dir/ThisWindow.Title AS file; |
|  | file.DFDBackend := ThisWindow.DFDBackend; |
|  | file.DDBackend := ThisWindow.DDBackend; |
|  | ThisWindow.IsChanged := TRUE; |
|  | CLOSE file; |
|  | } |
|  | Ans := TRUE; |
|  | } ELSE Ans := FALSE; |
|  | } |

## SavePrompt

|  |  |
| --- | --- |
|  | ALGORITHM SavePrompt() : BOOLEAN Ans { |
|  | OPEN SavePrompt dialog box AS sp; |
|  | Ans := ((sp.Output = “Yes” AND SaveFile()) OR sp.Output = “No”); |
|  | } |

## SavePromptDecompose

|  |  |
| --- | --- |
|  | ALGORITHM SavePromptDecompose() : BOOLEAN Ans { |
|  | OPEN SavePrompt dialog box AS sp; |
|  | Ans := ((sp.Output = “Yes” AND SaveDecompose()) OR sp.Output = “No”); |
|  | } |

## UpdateConnects

|  |  |
| --- | --- |
|  | ALGORITHM UpdateConnects(DATAFLOW df) : VOID { |
|  | s1 := df.Start.Shape; |
|  | s2 := df.End.Shape; |
|  | IF (s1.Type = DataProcess) { |
|  | IF (s2.Type = DataProcess) { |
|  | ADD df IN ThisWindow.DFDBackend.Connects[(s1, s2)]; |
|  | ADD df IN ThisWindow.DFDBackend.Connects[(s2, s1)]; |
|  | } ELSE { |
|  | ADD df IN ThisWindow.DFDBackend.Connects[(s1, s2)]; |
|  | FOR TWOPATH key in ThisWindow.DFDBackend.Connects.keys() { |
|  | IF ( |
|  | key.Uedge = s2 AND |
|  | ThisWindow.DFDBackend.Connects[key].size > 0 |
|  | ) { |
|  | FOR ( |
|  | PATH \_ |
|  | ) IN ThisWindow.DFDBackend.Connects[key] { |
|  | ADD ( |
|  | NEW TWOPATH(df, key) |
|  | ) IN ( |
|  | ThisWindow.DFDBackend.Connects[ |
|  | (s1, key.Vedge) |
|  | ] |
|  | ); |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } ELSE { |
|  | ADD df IN ThisWindow.DFDBackend.Connects[(s1, s2)]; |
|  | FOR TWOPATH key in ThisWindow.DFDBackend.Connects.keys() { |
|  | IF ( |
|  | key.Vedge = s1 AND |
|  | ThisWindow.DFDBackend.Connects[key].size > 0 |
|  | ) { |
|  | FOR PATH \_ IN ThisWindow.DFDBackend.Connects[key] { |
|  | ADD ( |
|  | NEW TWOPATH(key, df) |
|  | ) IN ( |
|  | ThisWindow.DFDBackend.Connects[ |
|  | (key.Uedge, s2) |
|  | ] |
|  | ); |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |

## ValPositions

|  |  |
| --- | --- |
|  | ALGORITHM ValPositions(POSITIONS P) : BOOLEAN Ans { |
|  | wp := ThisWindow.Workspace.Positions; |
|  | ans := ALL( |
|  | ( |
|  | (p[0] >= wp[0][0] AND p[0] <= wp[1][0]) AND |
|  | (p[1] >= wp[0][1] AND p[1] <= wp[1][1]) |
|  | ) FOR POSITION p IN P |
|  | ); |
|  | } |