Pseudocode

# Functions

* ChooseFile: Opens a fileChooser with extension as parameter, to be appended in the file name, returns Directory and file name
* ThisWindow: Current Window

# Structures

## DFDBACKEND

DFD Backend

* Connects (MAP of POSITIONS to PATHS) : Connections beween two data entities
* Data (JSON file): Object properties
* UndirGraph (GRAPH): Undirected graph of shapes
* DataFlowArray (ARRAY of INTEGERS): Stores which numers have been used for system names for existing dataflows

## EDGE

Edge

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

## FILE

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

## PATH

Path

* Type (ENUM): SHAPE, TwoPath
* Union
* OneWay (SHAPE) : Direct path
* TwoWay (TWOPATH) : Indirect path

## PATHS

Array of PATH

## POSITION

Array of 2 INTEGER: x, y

## POSITIONS

Set of 2 POSITION : Upper left, Lower right

## SHAPE

Shape

* Type (ENUM): Data entity, Data flow, Data store, External Entity, Label
* Position (POSITIONS) : Position of shape

## SHAPEANCHOR

Anchor point of Shape

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

## 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
* DFCounts (SET OF INTEGERS) : List of counts used for denerating system names of existing data flows

## TWOPATH

Path of length 2

* Uedge (SHAPE) : Edge outgoing from data entry
* Vedge (SHAPE) : Edge incoming to data entry

# Algorithms

## DebugFile

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

## DebugIsolated

|  |  |
| --- | --- |
|  | ALGORITHM DebugIsolated() : BOOLEAN Ans { |
|  | g := ThisWindow.DFDBackend.UndirGraph; |
|  | n := g.n; |
|  | IF (n == 0) Ans := FALSE; ELSE { |
|  | isVisited := ARRAY(n) {FALSE, …}; |
|  | isVisited[0] := 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);) |
|  | } |
|  | Ans := ANY(NOT x FOR BOOLEAN x IN isVisited); |
|  | } |
|  | } |

## DebugUnnecessary

|  |  |
| --- | --- |
|  | ALGORITHM DebugUnnecessary() : BOOLEAN Ans { |
|  | mc := ThisWindow.DFDBackend.Connects; |
|  | Ans := ANY(mc[x].length() > 1 FOR x in mc.keys()); |
|  | } |

## Exit

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

## 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) { |
|  | 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); |
|  | } |

## InsertDataFlow

|  |  |
| --- | --- |
|  | ALGORITHM InsertDataFlow(POSITIONS P) : VOID { |
|  | df := NEW DATAFLOW; |
|  | IF (ValPositions(P)) { |
|  | df.Start := FindShapeAnchor(P[0]); |
|  | IF (df.Start NOT NULL) { |
|  | df.End := FindShapeAnchor(P[1]); |
|  | IF (df.End NOT NULL) { |
|  | df.Name := SmallestName(Dataflow); |
|  | df.updateConnects(); |
|  | df.DDWizard(); |
|  | ADD df IN ThisWindow.DFDBackend.Data; |
|  | DISPLAY ( |
|  | ThisWindow.DFDBackend.Data |
|  | ) IN ThisWindow.Workspace; |
|  | } |
|  | } |
|  | } |
|  | } |

## MoveShape

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

## 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”; |
|  | } |
|  | } |

## RenameShape

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

## ResizeShape

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

## 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; |
|  | 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; |
|  | } |

## 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”); |
|  | } |

## 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 |
|  | ); |
|  | } |