Summer Fellowship Report

on

Chemical PFD Tool

Sumit Sahu

Under the guidance of

Prof. Kannan Moudgalya

Department of Chemical Engineering

FOSSEE, Python
IIT Bombay

Acknowledgement

I would like to take this opportunity to express my greatest gratitude to mentor, Mr. Pravin Kumar Dalve for guiding, supporting and helping me in every possible way. I was extremely fortunate to have him as my mentor as he provided insightful solutions to problems faced by us thus contributing immensely towards the completion of this project. I would like to thank FOSSEE Team and IIT Bombay for giving me this opportunity and providing a platform to exhibit our skills. I would also like to express our deepest gratitude to the faculty of my collage for giving me an opportunity to be a part of this fellowship.

Contents

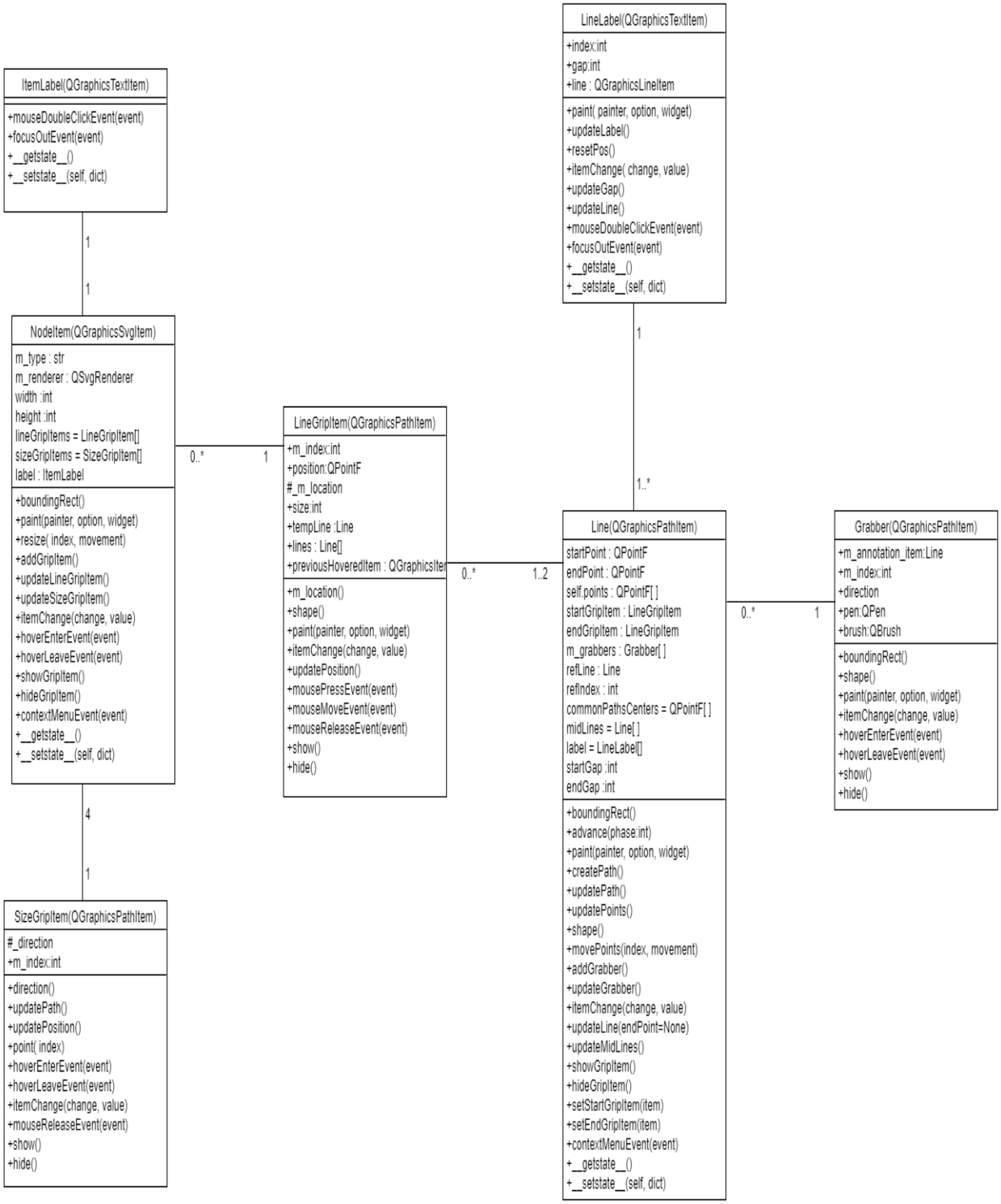
Chapter 1

Introduction

In Chemical process flow diagram tool we need components and lines connecting them. Component are created as QGraphicsSvgItem passing svg file of component.

1.1 Approach

We have used the python language to create and system involving an object-oriented approach to each of the components, their connections and defining properties of each of them. Our primary graphical view and UI was built using PyQt5 which is a python wrapper library over the C/C++ Qt framework. The GUI empowers user to select component from a wide database by searching its name . Component is rendered on canvas using svg(scable vector graphics) files. For connecting lines between different components, grips are provided to each component according to their shape.User can connect line by simply connecting grips.If connecting part is line then user grip is rectangular else circular.


ItemLabel(QGraphicsTextItem)
+mouseDoubleClickEvent(event)
+focusOutEvent(event)
+__getstate__()
+__setstate__(self, dict)
1
1
NodeItem(QGraphicsSvgItem)
m_type : str
m_renderer : QSvgRenderer
width :int
height :int
lineGripItems = LineGripItem[]
sizeGripItems = SizeGripItem[]
label : ItemLabel
+boundingRect()
+paint(painter, option, widget)
+resize(index, movement)
+addGripItem()
+updateLineGripItem()
+updateSizeGripItem()
+itemChange(change, value)
+hoverEnterEvent(event)
+hoverLeaveEvent(event)
+showGripItem()
+hideGripItem()
+contextMenuEvent(event)
+__getstate__()
+__setstate__(self, dict)
0..*
1
LineGripItem(QGraphicsPathItem)
+m_index:int
+position:QPointF
#_m_location
+size:int
+tempLine :Line
+lines : Line[]
+previousHoveredItem : QGraphicsIter
+m_location()
+shape()
+paint(painter, option, widget)
+itemChange(change, value)
+updatePosition()
+mousePressEvent(event)
+mouseMoveEvent(event)
+mouseReleaseEvent(event)
+show()
+hide()
0..*
1..2
LineLabel(QGraphicsTextItem)
+index:int
+gap:int
+line : QGraphicsLineItem
+paint(painter, option, widget)
+updateLabel()
+resetPos()
+itemChange(change, value)
+updateGap()
+updateLine()
+mouseDoubleClickEvent(event)
+focusOutEvent(event)
+__getstate__()
+__setstate__(self, dict)
1
1..*
Line(QGraphicsPathItem)
startPoint : QPointF
endPoint : QPointF
self.points : QPointF[ ]
startGripItem : LineGripItem
endGripItem : LineGripItem
m_grabbers : Grabber[ ]
refLine : Line
refIndex : int
commonPathsCenters = QPointF[ ]
midLines = Line[ ]
label = LineLabel[]
startGap :int
endGap :int
+boundingRect()
+advance(phase:int)
+paint(painter, option, widget)
+createPath()
+updatePath()
+updatePoints()
+shape()
+movePoints(index, movement)
+addGrabber()
+updateGrabber()
+itemChange(change, value)
+updateLine(endPoint=None)
+updateMidLines()
+showGripItem()
+hideGripItem()
+setStartGripItem(item)
+setEndGripItem(item)
+contextMenuEvent(event)
+__getstate__()
+__setstate__(self, dict)
0..*
1
Grabber(QGraphicsPathItem)
+m_annotation_item:Line
+m_index:int
+direction
+pen:QPen
+brush:QBrush
+boundingRect()
+shape()
+paint(painter, option, widget)
+itemChange(change, value)
+hoverEnterEvent(event)
+hoverLeaveEvent(event)
+show()
+hide()
4
1
SizeGripItem(QGraphicsPathItem)
#_direction
+m_index:int
+direction()
+updatePath()
+updatePosition()
+point(index)
+hoverEnterEvent(event)
+hoverLeaveEvent(event)
+itemChange(change, value)
+mouseReleaseEvent(event)
+show()
+hide()

Chapter 2

Technology Stack

The technologies that we used for the development of the GUI Software.Following are the technology stack we have use:

2.1 Python

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace.

2.2 PyQt

PyQt is a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in. PyQt is free software developed by the British firm Riverbank Computing. We used Latest version of PyQt i.e PyQt5. PyQt5 is a comprehensive set of Python bindings for Qt v5. It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android.

2.3 Fbs

Fbs lets you use Python and Qt to create desktop applications for Windows, Mac and Linux. It can create installers for your app, and automatically handles the packaging of third-party libraries and data files. These things normally take weeks to figure out. fbs gives them to you in minutes instead.

2.4 Git

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows. We Used Github for the Git Version Control System.

Chapter 3

Implementation

3.1 Unit Operations

Create a separate class for each component inheriting class NodeItem which takes a string which is path of svg(scable vector graphics) file of component. Whenever new component is created an object of NodeItem is created,similar for removing .

3.2 Connecting Line

Line is a QGraphicsPathItem ,path of which is combination of horizontal and vertical line alternatively, path is created using points stored in line itself .Whenever a line is created it's algorithm sets its initial path such that it doesn't intersects with item to which it joins.

3.3 Line Grip

LIne Grip are child of component and used for connecting line. Relative position of line grip is stored in component class. When a component is created all line grips attached to it gets created.

3.4 Size Grip

Size grip are child of component and used for resizing component. When a component is created all size grips attached to it gets created.

3.5 Text Label

Text label are QGraphicsTextItem, on mouse double click it becomes editable else movable.User can add text lable to component and line.

Interaction with component includes:

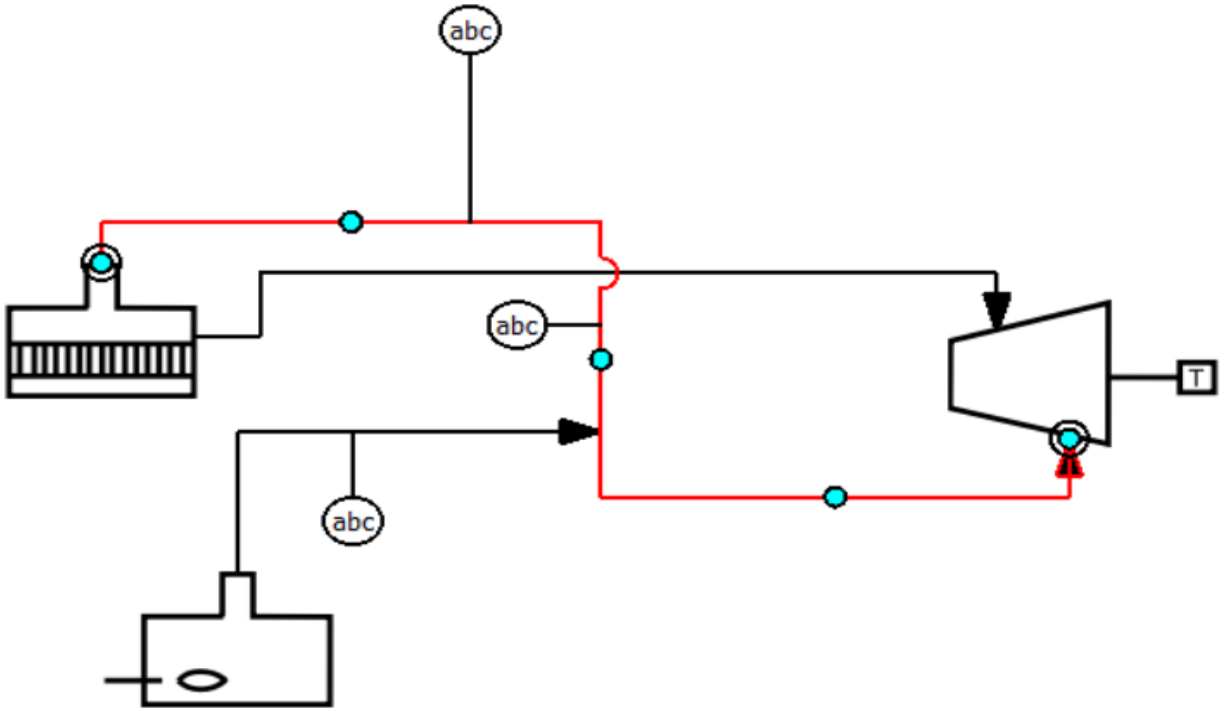
- Add new component
- Removing a component
- Making the connections between each component
- resizing component
- Making the connections between component and line
- Add text label to line and component
- Moving the text label

Chapter 4

Features

Following are the main features of the GUI application:

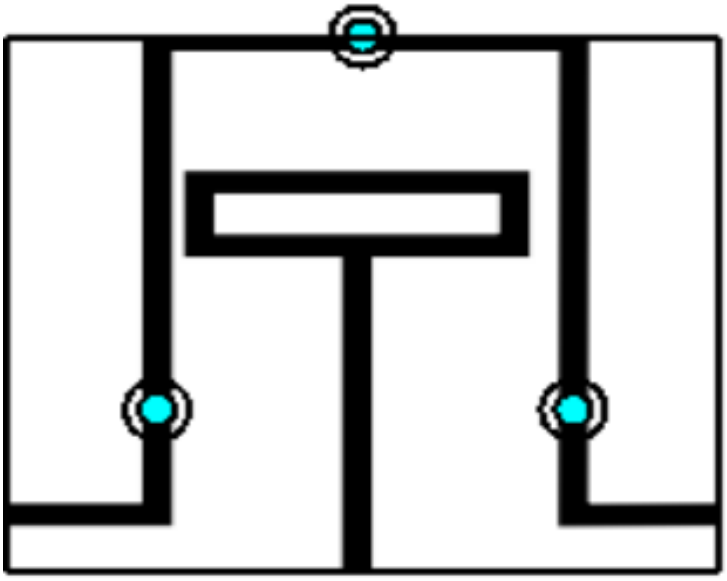
4.1 Feature of line



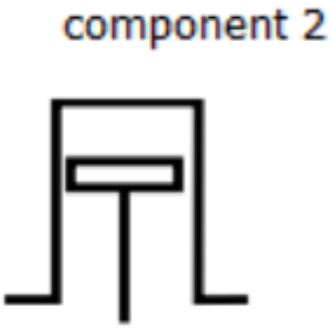
- A segment of line can be selected and moved to adjust line
- Arrow indicates the direction of chemical process

- Upper line has curve in case of intersecting line
- A line can be connected in mid of line
- Line gets updated when it's start or end points component moves
- Multiple text label can be added to line
- When selected it gets highlighted
- Text label can move on segment on which they are added

4.2 Feature of Component



component 1



- Add and remove from canvas
- Resize component in any direction
- Has grips to connect line

- When removed lines and labels connected to it gets removed
- Component is movable
- Add movable text label to component

4.3 Canvas

Canvas is the area where user can draw the required flowsheet. User can con net multiple components with each other by connecting the nodes on the different components with a line. The canvas can be zoomed in or Out.

Chapter 5

To Do

5.1 Line starting from line

In current implementation a connecting line can start from line grip item and end on line grip or another line. For a pfd it is required to start a line from mid of another line.

5.2 Preparing class entries

Class Definition

Under src/main/python/shapes/shape.py, using the following as an example, one can create his own class definition for the symbol. The grip list is the percentage position of the grip item object along with the parent's width and height, the third value is its position and the fourth value if specified is the width/height if the grip is a line grip item.

```
class HorizontalVessel(NodeItem):
    def __init__(self):
        super(HorizontalVessel, self).__init__("svg/Process Vessels/Horizontal Vessel")
        self.grips = [
            [50, 100, "top", 87.08554680344],
            [0, 50, "left"],
            [100, 50, "right"],
            [50, 0, "bottom", 87.08554680344]
        ]
```

Self.grip is list of grips for connecting line.List of grip having size = 3 are circle else rectangular.