# NSI Terminale Spé — Logiciel d'impresion d'un dessin sur une machine AxiDraw V3

PLE Jules IRDEL Jb

January 2026

## 1 Problématique :

Comment mettre en place tout les objets pour controler une machine de dessin en réseau ?

## 2 Introduction :

Ce programme permet à l'utilisateur de dessiner des formes sur une interface graphique. Il peut envoyer les formes dessinées sur l'ordinateur à une machine AxiDraw par reseau qui elle va dessiner les formes envoyées par l'utilisateur.

## 3 Aspect théorique :

Le projet repose sur plusieurs notions théoriques importantes :

-Robot d'écriture : l'AxiDraw est un traceur qui se déplace sur deux axes (X et Y) et permet de dessiner des formes précises à partir de coordonnées.

-Graphisme vectoriel (SVG) : contrairement aux images matricielles, le SVG décrit les formes à l'aide de formules mathématiques (lignes, rectangles, ellipses), ce qui est idéal pour le pilotage d'une machine.

Architecture client / serveur :

-le client est l'application de dessin sur l'ordinateur de l'utilisateur

-le serveur reçoit le fichier SVG et le transmet à la machine AxiDraw.

## 4 Part sociétale :

D'un point de vue sociétal, ce type de logiciel rend accessible la création graphique analogique à un public plus large.

Il permet par exemple :

-de réaliser des affiches personnalisées

-de faire de la calligraphie automatisée

-de produire des dessins précis sans compétences artistiques avancées.

# 5 Partie algorithmique:

## 5.1 Pour le réseau :

Pour le client :

- Fonction imprimer_svg_client qui prend en entrée le svg et qui l'envoie au serveur et retourne le message envoyé

Pour le serveur :

- Fonction index instanciant le site web
- Fonction recevoir, vérifiant la présence d'un fichier svg afin de l'enregister puis de l'imprimer et retourne le message

## 5.2 Pour le logiciel :

- Fonction start_draw qui permet d'enregistrer les coordonnées du premier point de la forme ou le client a cliqué.
- Fonction draw qui elle permet de montrer a quoi va resembler la forme en train d'etre dessinée tout cela pendant que le client maintien le clic.
- Fonction stop_draw sert quand le clic est relaché a créer la forme faite par le client.
- La fonction save_co sert a enregistrer les informations de la forme faite comme ces coordonnées, quelle forme etait faite et son identifiant qui sert a unifié chaques formes crées.
- La fonction convertion nous permet d'enregistrer toutes les formes du canvas en code svg.
- La fonction imprimer_svg_client sert à envoyer le code svg du canvas au serveur qui se chargera de l'imprimer.
- Fonction make_button permet de créer un bouton.

# 6 Ressources documentaire :

-GitHub (documentation et exemples de projets similaires)
-Documentation officielle de Tkinter
-Documentation SVG (W3C)
-Documentation de la machine AxiDraw
-Stack Overflow pour utiliser os
-Documentation Cherrypy pour utiliser cherrypy
https://stackoverflow.com/questions/71104397/how-to-convert-svg-string-to-svg-file-using-python

https://www.datacamp.com/fr/tutorial/how-to-check-if-a-file-exists-in-python
https://docs.cherrypy.dev/en/latest/_modules/cherrypy/tutorial/tut09_files.html#FileDemo.download
Source - https://stackoverflow.com/questions/38991286/cherrypy-upload-file
https://www.datacamp.com/fr/tutorial/comprehensive-tutorial-on-using-pathlib-in-python-for-file-system-manipulation

- Utilisation Grok et ChatGPT : n'a pas abouti pour le réseau et le logiciel

# 7 Part projective et part minimale:

- Part minimale:
  - développement d'une application de dessin convertissant en svg
  - développement d'un client et d'un serveur
- Part projective:
  - Etendre le logiciel a d'autre formes
  - imprimer des formes Latex
  - étendre le logiciel de dessin à des formes géométriques plus complexes
  - étendre la connexion réseau pour internet

# 8 Outils logiciel :

- Tkinter : Bibliotheque qui permet de gerer l'intervace graphique du projet
- AxiDraw : Permet de gerer le moteur de la machine
- os : pour le traitement des fichiers
- Requests : pour l'envoi en réseau

Installation de Axidraw :
installer le prebuilt : `>> sudo pip install "nom du fichier zip"`
– dezipper les fichiers –
installer axidraw : `>> sudo python setup.py install`

# 9 Matériels nécessaires :

- 2 ordinateurs (client + serveur)
- Une machine AxiDraw V3

# 10 Méthodologie :

-Analyse du fonctionnement d'AxiDraw.
-Création de l'interface graphique de dessin.
-Enregistrement des formes et de leurs coordonnées.
-Conversion du dessin en SVG.
-Mise en place de la communication client / serveur.
-Tests et corrections du programme.

## 10.1 Voici le fonctionnement en détail :

-Gestion du dessin

- Récupération des coordonnées de la souris.
- Création des formes (ligne, rectangle, ellipse).
- Sauvegarde des coordonnées dans une structure de données (liste de dictionnaires).
- Conversion en SVG
- Transformation des coordonnées du canvas en coordonnées SVG.
- Application d'un facteur d'échelle pour adapter le dessin aux dimensions physiques de l'AxiDraw.

-Communication réseau

- Envoi du code SVG au serveur via une requête HTTP (POST).
- Le serveur se charge ensuite d'interpréter le SVG et de piloter la machine.

# 11 Repartition des tâches au sein du groupe :

Nous avons essayés répartir les tâches équitablement lors de ce projet. Nous avons répartis ce projet en deux. Jean-Batiste c'est occupé de faire la partie reseau qui est donc que l'on peut envoyer notre dessin à distance puis Jules c'est occupé de faire la partie d'intervace graphique plus la transformation en code svg du dessin, cette partie consistait donc de créer un interface graphique où l'on puisse dessiner des formes puis les convertir en code svg pour ensuite les faires dessiner par la machine.

# 12 Journal de Bord :

## 12.1 IRDEL Jb :

### 12.1.1 Octobre :

- 2 sem oct. : début du projet, documentation
- 3 sem oct. : tentative d'installation d'axidraw
- 4 sem oct. : installation d'axidraw avec le terminal

### 12.1.2 Novembre :

- 1 sem nov. : utilisation du serveur connecté à l'imprimante, installation des dépendances
- 2 sem nov. : création d'un site et transfert d'un premier fichier svg du client au serveur via index/download
- 3 sem nov. : utilisation de la méthode POST pour l'interaction entre le client et serveur, premières impressions
- 4 sem nov. : création de deux programmes, un pour le client et un autre pour le serveur avec cherrypy

### 12.1.3 Decembre :

- 1 sem dec. : vérification de la compatiblité entre le programme de Jb et de Jules, configuration de la bonne taille, A4 pour que le robot ai des limites
- 2 sem dec. : tentative d'intégration de Latex

## 12.2 PLE Jules:

### 12.2.1 Octobre :

Création d'un programme avec un menu qui me permetais d'imprimer les formes soit d'une taille déjà donnée ou alors que je choisisais moi même. Le but de ce programme était de comprendre la bibliotheque AxiDraw pour voir comment les commandes marchaient et comment la machine reagisait aux commandes. Je n'ai pas utilisé ce programme pour mon projet car il ne me paraiser pas utile, il etait pour moi juste dans un but de comprendre la bibliotheque AxiDraw.

### 12.2.2 Novembre :

Création du projet de créer un paint simple avec quelque formes avec une intervace graphique. Cela m'a prit deux semaines environ.

- Apres ca je l'ai adapter pour notre projet de sauvgarder les coordonnées des formes. Pendant la semaine du 17/11
- J'ai fais pour pouvoir convertir en svg avec la fonction "convertion" dans la semaine du 27/11

#### 12.2.3 Decembre :

- Creation de la fonction Clear_Last le 2/12
- Import de la fonction qui permet d'envoyer au serv le 4/12

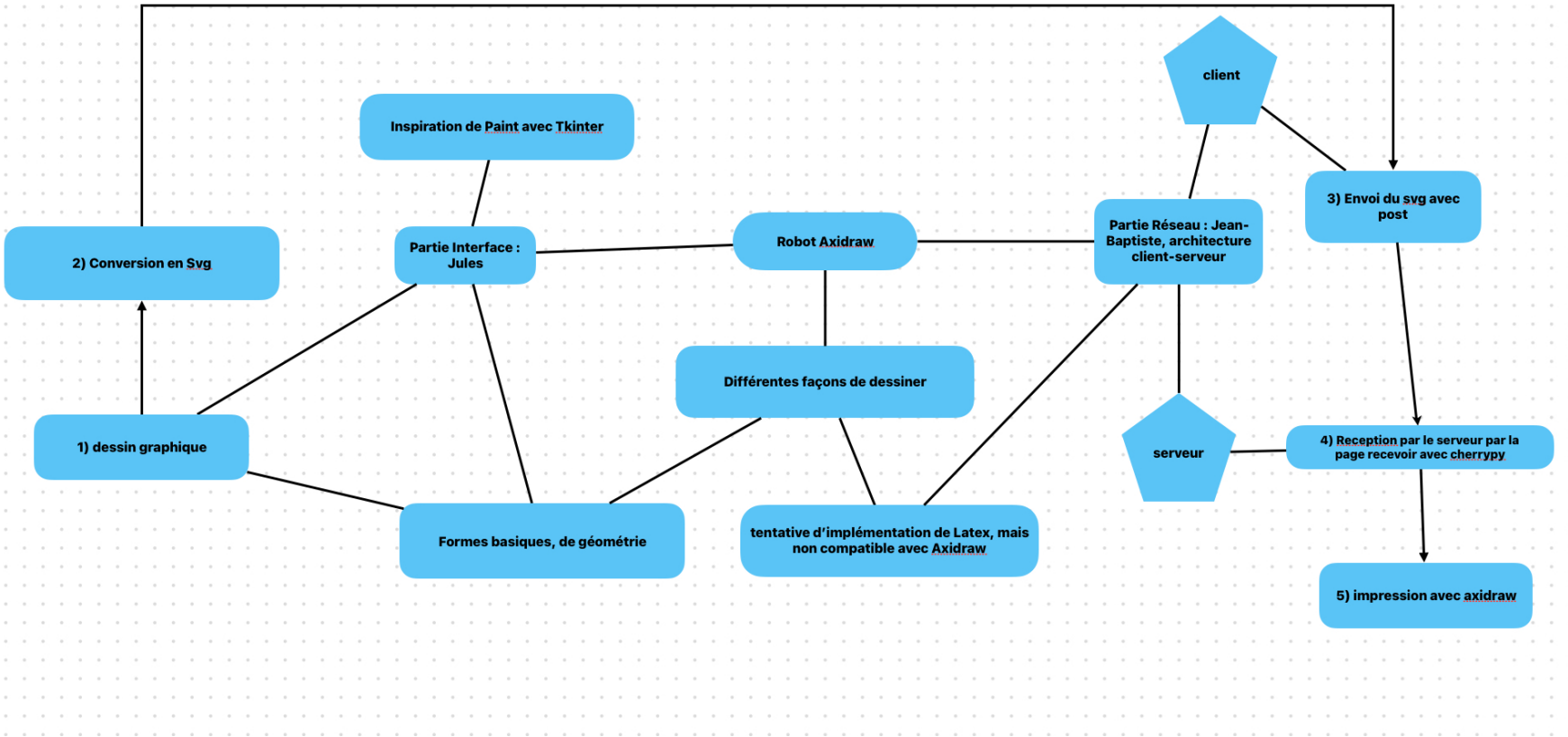


Figure 1: Mind Map du projet représentant l'architecture de notre système

# 13 Code

## 13.1 Client

Le code suivant permet d'envoyer sur le serveur le fichier svg.

```
# importation des biblioth ques
import requests

url = "http://172.16.100.30:5432/recevoir"

def imprimer_svg_client(svg,url):
        '''envoie sur le serveur le fichier svg pour
            imprimer'''

        # cr ation du dictionnaire "chaine" contenant le
            svg
        chaine = {'message' : svg}
        # transfert du svg au serveur
        response = requests.post(url, data = chaine)
        #  retourne   la reponse
        return response.text
```

## 13.2 Serveur

Le code suivant permet de gérer les pages cherrypy dans le but de recevoir le svg et de l'imprimer à l'aide d'axidraw

```
from pyaxidraw import axidraw
import cherrypy
import os

def index():
        ''' instanciation de la page index '''
        return "Hello"

def recevoir(message,imprimer=True):
        ''' instanciation de la page recevoir, permettant le
            traitement du fichier svg et son impression   '''
        # source pour le traitement du svg : https://
            stackoverflow.com/questions/71104397/how-to-
            convert-svg-string-to-svg-file-using-python
        with open("svgTest.svg", "w") as svg_file:
                        svg_file.write(message)
                        print("fichier enregistr ")
        # imprime le svg si imprimer = True
        if imprimer:
                ad = axidraw.AxiDraw()
                ad.plot_setup("svgTest.svg")
                ad.plot_run()
```

```
20
21          print(message)
22          # retourne le svg re u
23          return f"Chaine re ue : {message}"
24
25  # exposition des pages
26  index.exposed = True
27  recevoir.exposed = True
28
29  #  configuration   r seau
30  cherrypy.config.update({
31          'server.socket_host': '0.0.0.0',    # Accessible
                depuis tout le r seau
32          'server.socket_port': 5432,
33          'server.thread_pool': 8,}
34      )
35
36  #  lancement   du serveur web
37  cherrypy.tree.mount(index, "/")
38  cherrypy.tree.mount(recevoir, "/recevoir")
39
40  cherrypy.engine.start()
41  cherrypy.engine.block()
```

### 13.3 Interface

Le code suivant est l'interface de l'utilisateur similaire à Paint.

```
1   #Convertion en svg 27/11, Clear_Last 2/12, Import de la
        fonction qui permet d'envoyer au serv 4/12
2   # importation des biblioth ques
3   import requests
4   import tkinter as tk
5
6   # Configuration de la fen tre principale
7   root = tk.Tk()
8   root.title("Paint")
9   root.geometry("900x636")
10
11
12  # Variables globales
13  color = 'black'
14  size = 1
15  shape = ""
16  start_x, start_y = None, None
17  preview = None
18  saved_shapes = []
19  url = "http://172.16.100.30:5432/recevoir"
20
```

```
21 # --- Les fonctions principales ---
22
23 def set_shape(s):
24         global shape
25         shape = s
26         highlight_button(s)
27
28 def clear_canvas():#Clear tout le canvas
29         global saved_shapes
30         canvas.delete("all")
31         saved_shapes = []
32
33 def clear_last(event):#Clear la forme la plus proche de la
       souris quand un clique droit est effectu
34         global saved_shapes
35         x, y = event.x, event.y
36         cbl = canvas.find_closest(x, y)[0]
37         canvas.delete(cbl)
38         saved_shapes = [s for s in saved_shapes if s["iden"]
               != cbl]
39
40         print("Formes␣sup:", cbl)
41         print("Formes␣restantes:", saved_shapes)
42
43 def start_draw(event):
44         global start_x, start_y
45         start_x, start_y = event.x, event.y
46
47 def draw(event):
48         global preview
49         x, y = event.x, event.y
50
51         canvas.delete("preview")
52
53         if shape == 'ligne':
54                 preview = canvas.create_line(start_x,
                        start_y, x, y, fill=color, width=size,
                        dash=(4, 2), tags="preview")
55         elif shape == 'rectangle':
56                 preview = canvas.create_rectangle(start_x,
                        start_y, x, y, outline=color, width=size,
                         dash=(4, 2), tags="preview")
57         elif shape == 'oval':
58                 preview = canvas.create_oval(start_x,
                        start_y, x, y, outline=color, width=size,
                         dash=(4, 2), tags="preview")
59
60 def stop_draw(event):
61         global start_x, start_y, preview
62         x, y = event.x, event.y
```

```
63
64          canvas.delete("preview")
65
66          if shape == 'ligne':
67                  idd = canvas.create_line(start_x, start_y, x
                        , y, fill=color, width=size)
68                  save_co(idd,shape, start_x, start_y, x, y)
69          elif shape == 'rectangle':
70                  idd = canvas.create_rectangle(start_x,
                        start_y, x, y, outline=color, width=size)
71                  print(canvas.coords(idd))
72                  save_co(idd,shape, start_x, start_y, x, y)
73          elif shape == 'oval':
74                  idd = canvas.create_oval(start_x, start_y, x
                        , y, outline=color, width=size)
75                  save_co(idd,shape, start_x, start_y, x, y)
76
77          start_x, start_y, preview = None, None, None
78
79  def save_co(idd,forme, x1, y1, x2, y2): #Fonction qui sert
        enregistrer les coordonn es des figures
80          global saved_shapes
81          save = None
82          if forme == "rectangle":
83                  save = {"iden":idd,"shape": "rect", "x1": x1
                        , "y1": y1, "x2": x2, "y2": y2}
84          elif forme == "oval":
85                  save = {"iden":idd,"shape": "ellipse", "x1":
                         x1, "y1": y1, "x2": x2, "y2": y2}
86          elif forme == "ligne":
87                  save = {"iden":idd,"shape": "line", "x1": x1
                        , "y1": y1, "x2": x2, "y2": y2}
88          if save is not None:
89                  saved_shapes.append(save)
90          print(f"{saved_shapes}\n")
91          return saved_shapes
92
93  def convertion(sauv):
94          """Convertion␣des␣coordonn es␣des␣formes␣en␣code␣
                svg"""
95          conv = '␣<svg␣xmlns="http://www.w3.org/2000/svg"␣
                width="1200"␣height="800"␣>␣'
96          for form in sauv :
97                  x1 = form["x1"]
98                  x2 = form["x2"]
99                  y1 = form["y1"]
100                 y2 = form["y2"]
101                 echelle_x = 1.17
102                 echelle_y = 1.3
103                 if form["shape"] == "rect":
```

```
104                         w = x2 - x1
105                         h = y2 - y1
106                         conv = conv + f'<rect x="{x1 * 
                                echelle_x}" y="{y1 * echelle_y}" 
                                width="{w * echelle_x}" height="{
                                h * echelle_y}" style="stroke:
                                black;  fill:none;" />'
107                 elif form["shape"] == "ellipse":
108                         cx = (x1 + x2) / 2
109                         cy = (y1 + y2) / 2
110                         rx = (x2 - x1) / 2
111                         ry = (y2 - y1) / 2
112                         conv = conv +  f'<ellipse cx="{cx * 
                                echelle_x}" cy="{cy * echelle_y}" 
                                rx="{rx * echelle_x}" ry="{ry * 
                                echelle_y}" style="stroke:black; 
                                fill:none;"/>'
113                 elif form["shape"] == "line":
114                         conv = conv + f'<line x1="{x1 * 
                                echelle_x}" y1="{y1 * echelle_y}" 
                                x2="{x2 * echelle_x}" y2="{y2 * 
                                echelle_y}" style="stroke:black; 
                                stroke-width:2" />'
115         conv = conv + "</svg>"
116         return conv
117 
118 def imprimer_svg_client(svg,url):
119         '''envoie sur le serveur le fichier svg pour 
                imprimer'''
120 
121         # cr ation du dictionnaire "chaine" contenant le 
                svg
122         chaine = {'message' : svg}
123         # transfert du svg au serveur
124         response = requests.post(url, data = chaine)
125         #  retourne   la reponse
126         return response.text
127 
128 # --- Interface de l'utilisateur ---
129 toolbar = tk.Frame(root, bg="#ececec", height=50)
130 toolbar.pack(fill=tk.X, side=tk.TOP, pady=4)
131 
132 buttons = {}
133 
134 def make_button(parent, text, command, key):
135         b = tk.Button(parent, text=text, command=command, 
                relief=tk.RAISED, bg="#f2f2f2",\
136         activebackground="#dcdcdc", width=13, height=2, bd
                =3)
137         b.pack(side=tk.LEFT, padx=6, pady=6)
```

```
138         buttons[key] = b
139         return b
140
141 def highlight_button(active):
142         for key, btn in buttons.items():
143                 if key == active:
144                         btn.config(bg="#cce6ff", relief=tk.
                            SUNKEN)
145                 else:
146                         btn.config(bg="#f2f2f2", relief=tk.
                            RAISED)
147
148 make_button(toolbar, "Effacer", clear_canvas, "effacer") #
    Bouton Effacer
149 make_button(toolbar, "Ligne", lambda: set_shape('ligne'), "
    ligne") #Bouton pour faire une ligne
150 make_button(toolbar, "Rectangle", lambda: set_shape('
    rectangle'), "rectangle") #Bouton pour faire un rectangle
151 make_button(toolbar, "Oval", lambda: set_shape('oval'), "
    oval") #Bouton pour faire un oval
152 make_button(toolbar, "Creer␣Code␣SVG", lambda : convertion(
    saved_shapes) , "Code␣SVG")#Bouton pour cr er le code
    SVG
153 make_button(toolbar, "Envoyer␣au␣serveur", lambda :
    imprimer_svg_client(convertion(saved_shapes),url), "Envoi
    ")#Bouton pour envoyer au serv
154
155 # Le canvas principal
156 canvas = tk.Canvas(root, bg='white', highlightthickness=0,
    cursor="cross")
157 canvas.pack(fill=tk.BOTH, expand=True, padx=4, pady=4)
158
159 #    vnements
160 canvas.bind('<Button-1>', start_draw)
161 canvas.bind('<B1-Motion>', draw)
162 canvas.bind('<ButtonRelease-1>', stop_draw)
163 canvas.bind("<Button-3>", clear_last)
164
165 # Lancement
166 root.mainloop()
```