TAREA Herramienta CASE: Java desde UML con XML y CLIPS

En el documento adjunto **presentación.pdf** se indican las funciones de:

- La **aplicación web** formada por los archivos:
  UML.html
  script.js
- El **parser** del documento diagram.xmi generado por la aplicación web:
  Traductor.py
- Las plantillas, reglas y hechos para CLIPS generadas por **Traductor.py**:
  output.clp
- Finalmente se carga el archivo output.clp en **CLIPS** y se genera el código Java.

1 (5 Puntos) Unir en un solo programa Python, Traductor.py y la parte encargada de generar el código Java al ejecutar en CLIPS el archivo output.clp. Para ello usar la librería clipspy (ver documento EjemploCLIPSPy.pdf)

2 (3 Puntos) Modificar Traductor.py para que genere las plantillas, hechos, y reglas correspondientes a los otros tipos de relaciones (asociación bidireccional, composición y agregación). Para distinguir el código generado, los atributos de tamaño dinámico asociados a las relaciones de composición pueden codificarse usando ArrayList<> o TreeSet<> y para los de agregación LinkedList<>.

3 (2 Puntos) Lograr que la aplicación web desencadene la ejecución de `Traductor.py` cuando se genere el archivo `diagram.xmi`, puedes hacerlo utilizando Flask y combinando la funcionalidad de la aplicación web con la lógica de Python. Aquí está una guía paso a paso para implementar el apartado 3:

---

1. Actualizar la estructura del proyecto

Organiza los archivos del proyecto como sigue:

```
bash
Copiar código
/project
├── app.py                  # Archivo principal de Flask
├── Traductor.py            # Programa Python para traducir diagram.xmi
a output.clp
├── templates/
│   └── index.html          # HTML de la aplicación web
├── static/
│   ├── js/
│   │   └── scripts.js      # Lógica en JavaScript
│   └── css/
│       └── styles.css      # Estilos CSS
├── generated_files/
│   ├── diagram.xmi         # Salida generada por la aplicación web
```

```
│   └── output.clp          # Archivo generado por Traductor.py
└── requirements.txt        # Dependencias del proyecto
```

---

2. Integrar Flask con Traductor.py

En `app.py`, agrega una ruta específica que se encargue de procesar el archivo `diagram.xmi` y ejecutar `Traductor.py`. Este archivo se generará cuando el usuario presione el botón en la aplicación web.

`app.py`:

```
python
Copiar código
from flask import Flask, render_template, request, jsonify
import subprocess
import os

app = Flask(__name__)

# Ruta para servir la página principal
@app.route('/')
def home():
    return render_template('index.html')

# Ruta para manejar el archivo generado (diagram.xmi) y ejecutar
Traductor.py
@app.route('/process-diagram', methods=['POST'])
def process_diagram():
    try:
        # Verificar si el archivo existe
        diagram_path = os.path.join('generated_files', 'diagram.xmi')
        if not os.path.exists(diagram_path):
            return jsonify({'error': 'Archivo diagram.xmi no
encontrado'}), 400

        # Ejecutar Traductor.py
        subprocess.run(['python', 'Traductor.py'], check=True)

        # Confirmar éxito
        return jsonify({'message': 'Archivo procesado
correctamente'}), 200

    except subprocess.CalledProcessError as e:
        return jsonify({'error': f'Error al ejecutar Traductor.py:
{e}'}), 500
    except Exception as e:
        return jsonify({'error': f'Error inesperado: {e}'}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

---

3. Configurar el botón en el frontend

En `templates/index.html`, agrega un botón que realice una solicitud POST a la nueva ruta `/process-diagram` al generarse `diagram.xmi`.

index.html:

```
html
Copiar código
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>UML a Código Java</title>
    <script defer src="{{ url_for('static', filename='js/scripts.js')
}}"></script>
</head>
<body>
    <h1>Generador de Código Java desde UML</h1>
    <!-- Botón para generar diagram.xmi y procesar con Traductor.py --
>
    <button id="generate-diagram">Generar y Procesar Diagrama</button>
    <div id="status"></div>
</body>
</html>
```

---

4. Lógica en JavaScript para la solicitud POST

En `static/js/scripts.js`, escribe el código para enviar la solicitud POST al backend.

scripts.js:

```
javascript
Copiar código
document.getElementById('generate-diagram').addEventListener('click',
async () => {
    try {
        // Realizar la solicitud POST al servidor Flask
        const response = await fetch('/process-diagram', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
        });

        // Manejar la respuesta
        const result = await response.json();
        const statusDiv = document.getElementById('status');

        if (response.ok) {
            statusDiv.innerHTML = `<p style="color:
green;">${result.message}</p>`;
        } else {
            statusDiv.innerHTML = `<p style="color: red;">Error:
${result.error}</p>`;
        }
    } catch (error) {
        console.error('Error al procesar el diagrama:', error);
```

```
        document.getElementById('status').innerHTML = `<p
style="color: red;">Error inesperado: ${error.message}</p>`;
    }
});
```

---

5. Integrar `Traductor.py` con CLIPS y Clipspy

En `Traductor.py`, asegúrate de que al final de la generación del archivo `output.clp`, se invoque CLIPS utilizando la librería `clipspy`.

Ejemplo básico para ejecutar CLIPS con Clipspy:

```
python
Copiar código
from clips import Environment

def ejecutar_clips(clp_path):
    env = Environment()
    env.load(clp_path)
    env.run()

if __name__ == '__main__':
    # Ruta del archivo output.clp
    output_path = 'generated_files/output.clp'

    # Lógica del traductor aquí
    print(f"Traduciendo y generando: {output_path}")

    # Ejecutar CLIPS
    ejecutar_clips(output_path)
```

---

6. Prueba de la integración

1. Ejecuta `app.py`:

   ```
   bash
   Copiar código
   python app.py
   ```

2. Ve a `http://127.0.0.1:5000` en tu navegador.
3. Genera un diagrama UML y presiona el botón para procesarlo.