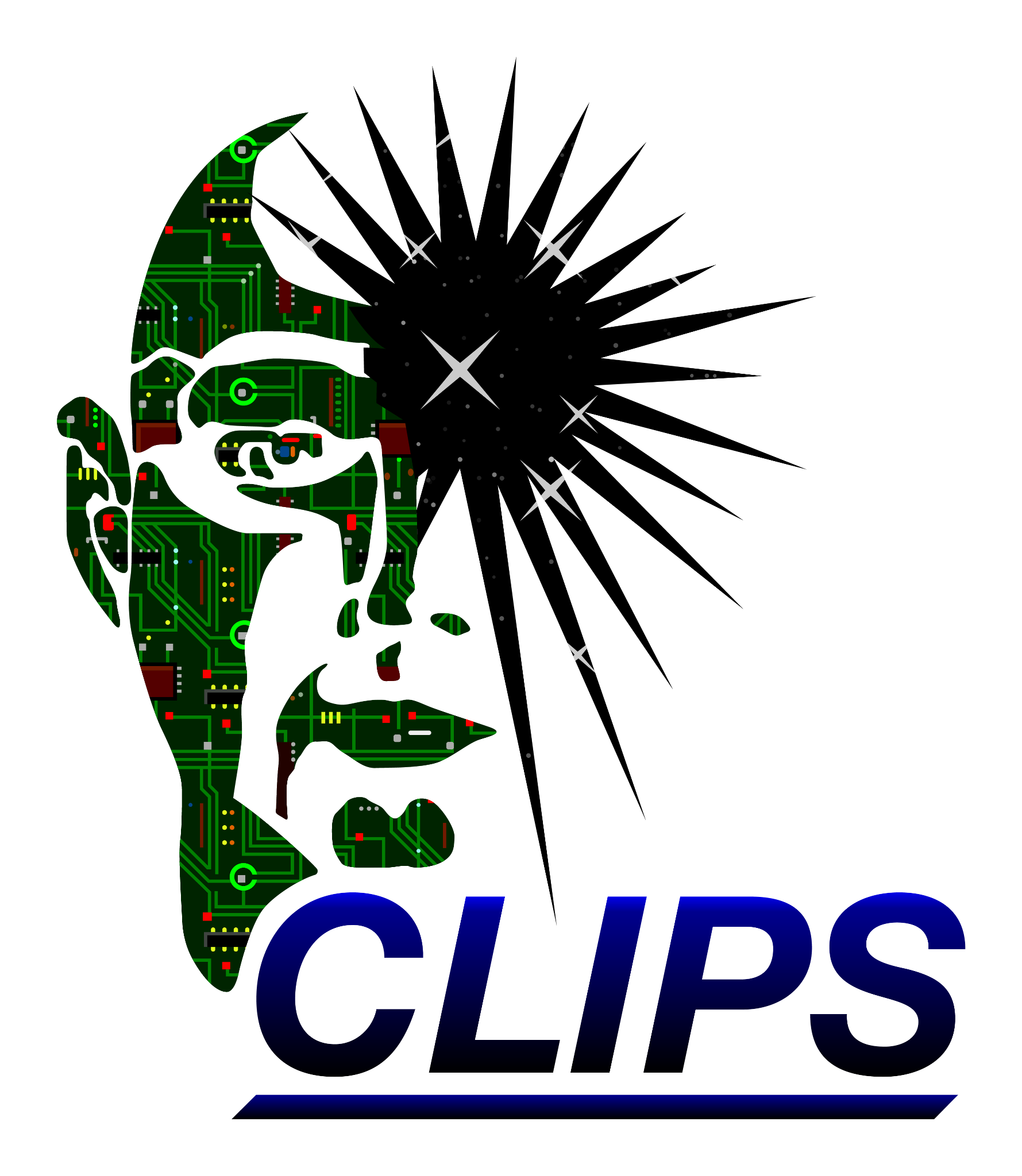
CLIPS - HERRAMIENTA CASE



Adrián Yared Armas de la Nuez

**Contenido**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

[**1. Apartado 1 3**](#_2c35568x6wg8)

[**1.1.1 Comando 3**](#_38dldri9aadx)

[**2. Apartado 2 4**](#_a2t2jddh3nhi)

[**2.1.1 asociación bidireccional 4**](#_m9z2e7lirwaa)

[**2.1.1.1 Código 5**](#_r558lq6acq3i)

[**2.1.1.2 Explicación del código 6**](#_tw4c5iegaema)

[**2.1.2 composición 6**](#_jm0ubtp8wb02)

[**2.1.2.1 Código 6**](#_jlmmwvw18589)

[**2.1.2.2 Explicación del código 7**](#_oep9f9ac9aa)

[**2.1.3 Agregación 7**](#_hq95phl7kqxo)

[**2.1.3.1 Código 7**](#_7i34g1uwp3oi)

[**2.1.3.2 Explicación del código 8**](#_r3jfnk63bot8)

[**3. Apartado 3 8**](#_xiu6sil1x56n)

[**3.1 Actualizar la estructura del proyecto 9**](#_t8cwbo6n4466)

[**3.2 Integrar Flask con Traductor.py 9**](#_uu80ts3sm1gw)

[**3.2.1 Comando 9**](#_1gjtfokf894)

[**3.3 Configurar el botón en el frontend 10**](#_528vfllt4wq1)

[**3.3.1 Comando XMI 10**](#_rvkwr5letuud)

[**3.3.2 Comando Clips 10**](#_s9c2rhjpvlbm)

[**3.3.3 Comando Java 11**](#_diiu2oicg55f)

[**3.4 Lógica en JavaScript para la solicitud POST 12**](#_kw6klf8ssxo7)

[**3.4.1 Comando 12**](#_v3bgy2345x9b)

[**3.5 Integrar Traductor.py con CLIPS y Clipspy 13**](#_81ja56arz83s)

[**2.5.1 Comando 13**](#_5q74weeeg1zz)

[**3.6 Prueba de la integración 14**](#_j9sx26vn2tjk)

[**3.6.1 Ejecuta app.py 14**](#_p89ensotymih)

[**3.6.2 Ve a http://127.0.0.1:5000 14**](#_n4cwwqgroack)

[**3.6.3 Genera un diagrama UML y presiona el botón para procesarlo. 14**](#_2ovl8dt6iq4a)

[**3.6.3.0.1 Descarga XMI 14**](#_oibgx62n9id3)

[**3.6.3.0.2 Descarga Clips 15**](#_dv6c5bp09emo)

[**3.6.3.0.3 Descarga Java 15**](#_xdi8pqpbm7ga)

[**3.6.3.1 Herencia 16**](#_n8cwkgrjt8hb)

[**3.6.3.1.1 Resultado XML 16**](#_i5z7koig6jfv)

[**3.6.3.1.2 Resultado Clips 16**](#_6gdvbep503js)

[**3.6.3.1.2 Resultado Java 17**](#_ti2ykipa1fv3)

[**3.6.3.2 Asociación 17**](#_xp9nyw4f8v77)

[**3.6.3.2.1 Resultado XML 17**](#_zdyyapc4paxs)

[**3.6.3.2.2 Resultado Clips 18**](#_2neu4hb1ia55)

[**3.6.3.2.3 Resultado Java 18**](#_vzojmlfwpzr6)

[**3.6.3.3 Asociación direccional 19**](#_jw4zb3xn1j7k)

[**3.6.3.3.1 Resultado XML 19**](#_nidqlnrev5f0)

[**3.6.3.3.2 Resultado Clips 19**](#_wvqmdak6ufwr)

[**3.6.3.3.3 Resultado Java 20**](#_pjbqwritp6vw)

[**3.6.3.4 Asociación Dependencia 21**](#_4w8486ujchyn)

[**3.6.3.4.1 Resultado XML 21**](#_ado3uy60whcf)

[**3.6.3.4.2 Resultado Clips 21**](#_90x7hbwwd6ut)

[**3.6.3.4.3 Resultado Java 21**](#_58oq7qqm5vd1)

[**3.6.3.5 Asociación Composición 22**](#_52ek9y37d570)

[**3.6.3.5.1 Resultado XML 22**](#_68mja87ar9jm)

[**3.6.3.5.2 Resultado Clips 23**](#_1eaoml4bwwu4)

[**3.6.3.5.3 Resultado Java 23**](#_ks1ha7kx6f42)

[**3.6.3.6 Asociación Agregación 24**](#_rsvo3xwb9ug6)

[**3.6.3.6.1 Resultado XML 24**](#_9tb1i19e65oo)

[**3.6.3.6.2 Resultado Clips 24**](#_cqo1nldskpum)

[**3.6.3.6.3 Resultado Java 25**](#_l6q7j3y7zm3s)

[**4. Proyecto en drive 26**](#_lmdngf8dlin3)

## 

## 

## 

## 

## **1. Apartado 1**

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)

#### **1.1.1 Comando**

Se añade todo el código del traductor.py en un solo archivo, que es el siguiente (app.py):

import os

import subprocess

from flask import Flask, render\_template, send\_file, request

from Traductor import process\_xmi\_file

app = Flask(\_\_name\_\_, static\_folder='static')

@app.route('/')

def index():

return render\_template('uml.html')

@app.route('/process\_xmi', methods=['POST'])

def process\_xmi():

xmi\_file = request.files['xmi']

if not xmi\_file:

return "No se envió ningún archivo XMI", 400

input\_file = 'static/temp/input.xmi'

output\_file = 'static/temp/output.clp'

xmi\_file.save(input\_file)

process\_xmi\_file(input\_file, output\_file)

try:

return send\_file(output\_file, as\_attachment=True)

except FileNotFoundError:

return "Error al generar el archivo CLIPS", 500

@app.route('/process\_clips\_to\_java', methods=['POST'])

def process\_clips\_to\_java():

xmi\_file = request.files['xmi']

if not xmi\_file:

return "No se envió ningún archivo XMI", 400

input\_file = 'static/temp/input.xmi'

output\_file = 'static/temp/output.clp'

java\_file = 'static/temp/output.java'

xmi\_file.save(input\_file)

process\_xmi\_file(input\_file, output\_file)

try:

subprocess.run(['python', 'clipsToJava.py', output\_file], check=True)

if os.path.exists(java\_file):

return send\_file(java\_file, as\_attachment=True)

else:

return "Error: El archivo Java no fue generado.", 500

except subprocess.CalledProcessError as e:

return f"Error al ejecutar clipsToJava.py: {e}", 500

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

Comando de salida del archivo como output.clp

output\_file = 'static/temp/output.clp'

## **2. Apartado 2**

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<>.

#### **2.1.1 asociación bidireccional**

El código de la relación bidireccional se encuentra en la función extract\_directed\_associations(). Este código extrae asociaciones dirigidas (unidireccionales) y las convierte en asociaciones bidireccionales en las clases correspondientes. Para implementar esto, el código actual ya asigna atributos de lista a las clases para mantener las referencias mutuas.

##### **2.1.1.1 Código**

def extract\_directed\_associations(root, class\_dict):

obj\_id = 1

directed\_associations = []

for elem in root.findall('.//packagedElement'):

type\_attr = elem.get('{http://schema.omg.org/spec/XMI/2.1}type')

if type\_attr == 'uml:DirectedAssociation':

member\_end = elem.get('memberEnd')

if member\_end:

source, target = member\_end.split()

owned\_ends = elem.findall('ownedEnd')

multiplicity\_source = None

multiplicity\_target = None

for owned\_end in owned\_ends:

end\_type = owned\_end.get('type')

if end\_type == source and multiplicity\_source is None:

multiplicity\_source = owned\_end.get('multiplicity1')

if end\_type == target and multiplicity\_target is None:

multiplicity\_target = owned\_end.get('multiplicity2')

if source and target:

directed\_associations.append({

'type': 'directedAssociation',

'source': source,

'target': target,

'multiplicity1': multiplicity\_source,

'multiplicity2': multiplicity\_target

})

# Añadir atributo en la clase source

class\_name = source

if class\_name in class\_dict:

if multiplicity\_target != "\*":

class\_dict[class\_name]['attributes'].append({

'name': f'{target.lower()}List{obj\_id}',

'visibility': 'private',

'type': f'{target}[]'

})

obj\_id += 1

else:

class\_dict[class\_name]['attributes'].append({

'name': f'{target.lower()}List{obj\_id}',

'visibility': 'private',

'type': f'HashSet<{target}>'

})

obj\_id += 1

return directed\_associations

##### **2.1.1.2 Explicación del código**

El código recorre todos los elementos <packagedElement> del archivo XMI para identificar las relaciones de tipo uml:DirectedAssociation. A continuación, asigna los atributos de las clases source y target con una lista o conjunto, según la multiplicidad definida. En la clase fuente (source), si la multiplicidad no es "", se agrega un atributo que representa una lista de objetos de la clase destino (target). Si la multiplicidad es "", se utiliza un conjunto (HashSet) para almacenar las referencias

#### **2.1.2 composición**

El código que maneja las composiciones está dentro de la función extract\_compositions(). En este código, se identifican las composiciones en el archivo XMI y se agregan como relaciones en el sistema.

##### **2.1.2.1 Código**

def extract\_compositions(root):

compositions = []

for elem in root.findall('.//packagedElement'):

type\_attr = elem.get('{http://schema.omg.org/spec/XMI/2.1}type')

if type\_attr == 'uml:Composition':

member\_end = elem.get('memberEnd')

if member\_end:

whole, part = member\_end.split()

owned\_ends = elem.findall('ownedEnd')

multiplicity\_target = None

for owned\_end in owned\_ends:

end\_type = owned\_end.get('type')

if end\_type == part:

multiplicity\_target = owned\_end.get('multiplicity')

if whole and part:

compositions.append({

'type': 'composition',

'whole': whole,

'part': part,

'multiplicity': multiplicity\_target

})

return compositions

##### **2.1.2.2 Explicación del código**

El código busca elementos de tipo uml:Composition dentro del archivo XMI. Al encontrar una composición, se extraen los elementos whole y part, que representan la clase que contiene al objeto y la clase que está contenida, respectivamente. A continuación, se añade una entrada para la composición, incluyendo la multiplicidad de los objetos contenidos (multiplicity\_target), lo que indica la cantidad de objetos que pueden estar asociados con el contenedor.

#### **2.1.3 Agregación**

La función extract\_aggregations() maneja la extracción de agregaciones. El código sigue un enfoque similar al de la composición, pero las relaciones de agregación no tienen la misma fuerte dependencia de vida.

##### **2.1.3.1 Código**

def extract\_aggregations(root):

aggregations = []

for elem in root.findall('.//packagedElement'):

type\_attr = elem.get('{http://schema.omg.org/spec/XMI/2.1}type')

if type\_attr == 'uml:Aggregation':

member\_end = elem.get('memberEnd')

if member\_end:

whole, part = member\_end.split()

owned\_ends = elem.findall('ownedEnd')

multiplicity\_target = None

for owned\_end in owned\_ends:

end\_type = owned\_end.get('type')

if end\_type == part:

multiplicity\_target = owned\_end.get('multiplicity')

if whole and part:

aggregations.append({

'type': 'aggregation',

'whole': whole,

'part': part,

'multiplicity': multiplicity\_target

})

return aggregations

##### **2.1.3.2 Explicación del código**

La función es prácticamente idéntica a la de la composición, pero en este caso, las relaciones extraídas son de tipo uml:Aggregation. El comportamiento es similar: se extraen las clases whole (contenedor) y part (contenida), junto con sus multiplicidades. La diferencia clave radica en que, a diferencia de la composición, la agregación no implica una dependencia fuerte entre los objetos contenidos.

## **3. Apartado 3**

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:

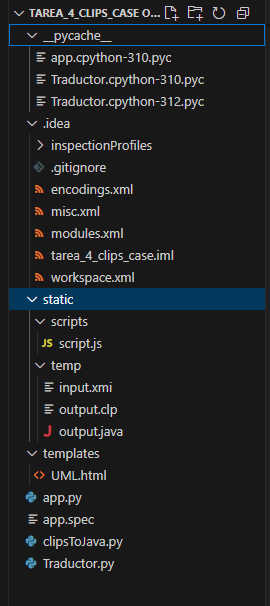
### 

### 

### 

### **3.1 Actualizar la estructura del proyecto**

**Cambio de estructura del proyecto:**



### **3.2 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.

#### **3.2.1 Comando**

Se importa flask:

from flask import Flask, render\_template, send\_file, request

Uso flask en para que se encargue de procesar el archivo xmi:

Esa línea crea una instancia de Flask y especifica que los archivos estáticos estarán en la carpeta static.

app = Flask(\_\_name\_\_, static\_folder='static')

### 

### **3.3 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.

#### **3.3.1 Comando XMI**

El botón:

<button onclick="downloadXMI()">Download XMI</button>

Llama a la función downloadXMI en scripts/scripts.js:

function downloadXMI() {

const xmi = generateXMI();

const blob = new Blob([xmi], { type: 'application/xml' });

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

a.download = 'diagram.xmi';

document.body.appendChild(a);

a.click();

document.body.removeChild(a);

}

Esto genera un archivo XMI en formato XML, lo convierte en un objeto Blob y crea un enlace de descarga temporal. Luego, simula un clic en el enlace para descargar el archivo y lo elimina del DOM.

#### **3.3.2 Comando Clips**

El botón:

<button onclick="downloadClips()">Descargar CLIPS</button>

Llama a la función downloadClips en scripts/scripts.js:

async function downloadClips() {

const xmi = generateXMI();

const blob = new Blob([xmi], { type: 'application/xml' });

const formData = new FormData();

formData.append('xmi', blob, 'diagram.xmi');

try {

const response = await fetch('/process\_xmi', {

method: 'POST',

body: formData,

});

if (!response.ok) {

throw new Error('Error al procesar el archivo XMI');

}

const clipsBlob = await response.blob();

const url = URL.createObjectURL(clipsBlob);

const a = document.createElement('a');

a.href = url;

a.download = 'diagram.clp';

document.body.appendChild(a);

a.click();

document.body.removeChild(a);

} catch (error) {

console.error('Error:', error);

alert('Ocurrió un error al descargar el archivo CLIPS.');

}

}

Esta función genera un archivo XMI, lo envía a un servidor para su procesamiento y luego descarga el resultado como un archivo CLIPS (.clp). Si ocurre un error, muestra un mensaje de alerta y lo registra en la consola.

#### **3.3.3 Comando Java**

El botón:

<button onclick="downloadJava()">Descargar Java</button>

Llama a la función downloadJava en scripts/scripts.js:

async function downloadJava() {

const xmi = generateXMI();

const blob = new Blob([xmi], { type: 'application/xml' });

const formData = new FormData();

formData.append('xmi', blob, 'diagram.xmi');

try {

const response = await fetch('/process\_clips\_to\_java', {

method: 'POST',

body: formData,

});

if (!response.ok) {

throw new Error('Error al procesar el archivo XMI');

}

const javaBlob = await response.blob();

const url = URL.createObjectURL(javaBlob);

const a = document.createElement('a');

a.href = url;

a.download = 'diagram.java';

document.body.appendChild(a);

a.click();

document.body.removeChild(a);

} catch (error) {

console.error('Error:', error);

}

}

La función downloadJava genera un archivo XMI, lo envía a un servidor para ser procesado y luego descarga un archivo Java generado a partir de la respuesta. Utiliza fetch para realizar la solicitud y Blob para manejar los archivos binarios.

### **3.4 Lógica en JavaScript para la solicitud POST**

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

#### **3.4.1 Comando**

La función downloadClips() genera un archivo XMI y lo envía mediante un POST a /process\_xmi. Si la respuesta es exitosa, recibe un archivo CLIPS (.clp), crea un enlace de descarga temporal y lo descarga automáticamente. Si ocurre un error, lo muestra en la consola y alerta al usuario.

async function downloadClips() {

const xmi = generateXMI();

const blob = new Blob([xmi], { type: 'application/xml' });

const formData = new FormData();

formData.append('xmi', blob, 'diagram.xmi');

try {

const response = await fetch('/process\_xmi', {

method: 'POST',

body: formData,

});

if (!response.ok) {

throw new Error('Error al procesar el archivo XMI');

}

const clipsBlob = await response.blob();

const url = URL.createObjectURL(clipsBlob);

const a = document.createElement('a');

a.href = url;

a.download = 'diagram.clp';

document.body.appendChild(a);

a.click();

document.body.removeChild(a);

} catch (error) {

console.error('Error:', error);

alert('Ocurrió un error al descargar el archivo CLIPS.');

}

}

### **3.5 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.

#### **2.5.1 Comando**

Import:

from clips import Environment

Al final de Traductor.py, se agrega una función que usa clipspy para cargar y ejecutar el archivo output.clp en un entorno CLIPS. Esto permite procesar las reglas automáticamente tras la generación del archivo, completando el flujo de transformación del XMI.

def run\_clips(output\_file):

env = Environment()

env.load(output\_file) # Carga el archivo output.clp en CLIPS

env.reset()

env.run() # Ejecuta las reglas en CLIPS

print("Ejecución de CLIPS completada.")

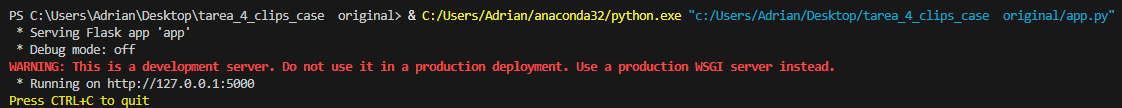
# Llamar a CLIPS después de generar output.clp

run\_clips(output\_file)

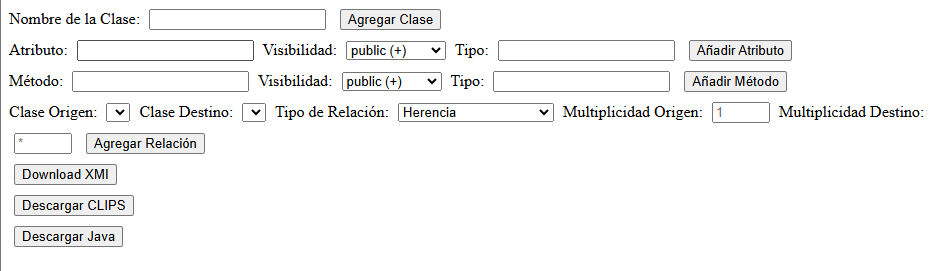
### **3.6 Prueba de la integración**

#### **3.6.1 Ejecuta app.py**

Al ejecutar, funciona sin problema y se ejecuta en 127.0.0.1:5000

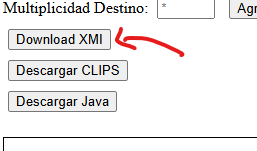


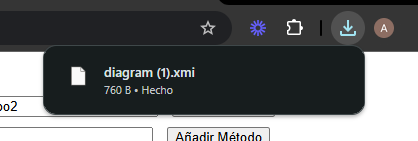
#### **3.6.2 Ve a** [**http://127.0.0.1:5000**](http://127.0.0.1:5000)

Al ir a la ruta se ve lo siguiente:  


#### **3.6.3 Genera un diagrama UML y presiona el botón para procesarlo.**

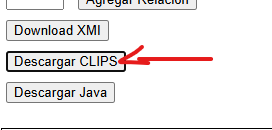
##### **3.6.3.0.1 Descarga XMI**

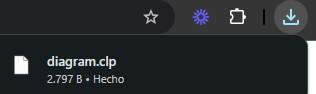




##### 

##### **3.6.3.0.2 Descarga Clips**

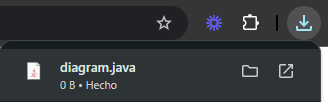




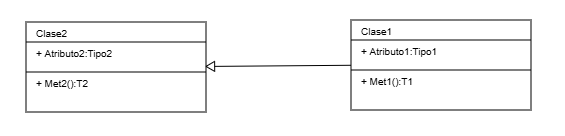
##### 

##### **3.6.3.0.3 Descarga Java**





##### **3.6.3.1 Herencia**



###### **3.6.3.1.1 Resultado XML**

<?xml version="1.0" encoding="UTF-8"?>

<XMI xmi.version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmlns:uml="http://www.omg.org/spec/UML/20090901">

<uml:Model xmi:type="uml:Model" name="UMLModel">

<packagedElement xmi:type="uml:Class" name="Clase1 ">

<ownedAttribute visibility="+" name="Atributo1" type="Tipo1" />

<ownedOperation visibility="+" name="Met1" type="T1" />

</packagedElement>

<packagedElement xmi:type="uml:Class" name="Clase2">

<ownedAttribute visibility="+" name="Atributo2" type="Tipo2" />

<ownedOperation visibility="+" name="Met2" type="T2" />

</packagedElement>

<packagedElement xmi:type="uml:Generalization" memberEnd="Clase1 Clase2">

</packagedElement>

</uml:Model>

</XMI>

###### **3.6.3.1.2 Resultado Clips**

(defclass Clase1

(is-a USER)

(slot Atributo1 (type SYMBOL))

(message-handler (Met1) (?self)

(bind ?resultado "Resultado de Met1")))

(defclass Clase2

(is-a Clase1) ;; Clase2 hereda de Clase1

(slot Atributo2 (type SYMBOL))

(message-handler (Met2) (?self)

(bind ?resultado "Resultado de Met2")))

;; Crear instancias de prueba

(definstances ejemplo

(instancia1 of Clase1 (Atributo1 valor1))

(instancia2 of Clase2 (Atributo1 valor1) (Atributo2 valor2)))

###### **3.6.3.1.2 Resultado Java**

// Java code for class Clase1

public class Clase1 extends Clase2 {

public Tipo1 Atributo1;

public T1 Met1() {

// method body

}

}

// Java code for class Clase2

public class Clase2 {

public Tipo2 Atributo2;

public T2 Met2() {

// method body

}

}

##### **3.6.3.2 Asociación**



###### **3.6.3.2.1 Resultado XML**

<?xml version="1.0" encoding="UTF-8"?>

<UMLModel>

<Class name="Clase1">

<Attribute name="Atributo1" type="Tipo1"/>

<Method name="met1" returnType="Tipo1"/>

</Class>

<Class name="Clase2">

<Attribute name="Atributo2" type="Tipo2"/>

<Method name="met2" returnType="Tipo2"/>

</Class>

<Association>

<End class="Clase1" multiplicity="1"/>

<End class="Clase2" multiplicity="\*"/>

</Association>

</UMLModel>

###### **3.6.3.2.2 Resultado Clips**

(deftemplate Clase1

(slot Atributo1))

(deftemplate Clase2

(slot Atributo2)

(slot RelacionClase1)) ;; Relación 1 a muchos con Clase1

(deffunction met1 (?instanciaClase1)

(bind ?valor (fact-slot-value ?instanciaClase1 Atributo1))

(return ?valor))

(deffunction met2 (?instanciaClase2)

(bind ?valor (fact-slot-value ?instanciaClase2 Atributo2))

(return ?valor))

(deffacts Instancias

(Clase1 (Atributo1 "Valor1"))

(Clase2 (Atributo2 "Valor2") (RelacionClase1 "ReferenciaClase1")))

###### **3.6.3.2.3 Resultado Java**

class Clase1 {

private Tipo1 atributo1;

public Clase1(Tipo1 atributo1) {

this.atributo1 = atributo1;

}

public Tipo1 met1() {

return atributo1;

}

}

class Clase2 {

private Tipo2 atributo2;

private Clase1 clase1; // Asociación con Clase1 (1 a muchos)

public Clase2(Tipo2 atributo2, Clase1 clase1) {

this.atributo2 = atributo2;

this.clase1 = clase1;

}

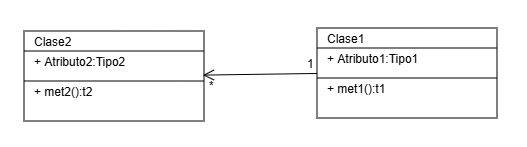
public Tipo2 met2() {

return atributo2;

}

}

##### **3.6.3.3 Asociación direccional**



###### **3.6.3.3.1 Resultado XML**

<?xml version="1.0" encoding="UTF-8"?>

<UMLModel>

<Class name="Clase1">

<Attribute name="Atributo1" type="Tipo1"/>

<Method name="met1" returnType="Tipo1"/>

</Class>

<Class name="Clase2">

<Attribute name="Atributo2" type="Tipo2"/>

<Method name="met2" returnType="Tipo2"/>

<Association name="RelacionClase2AClase1" direction="unidirectional">

<End class="Clase1" multiplicity="1"/>

<End class="Clase2" multiplicity="\*"/>

</Association>

</Class>

</UMLModel>

###### **3.6.3.3.2 Resultado Clips**

(deftemplate Clase1

(slot Atributo1))

(deftemplate Clase2

(slot Atributo2)

(multislot ReferenciasClase1)) ;; Relación unidireccional hacia Clase1

(deffunction met1 (?instanciaClase1)

(bind ?valor (fact-slot-value ?instanciaClase1 Atributo1))

(return ?valor))

(deffunction met2 (?instanciaClase2)

(bind ?valor (fact-slot-value ?instanciaClase2 Atributo2))

(return ?valor))

(deffacts Instancias

(Clase1 (Atributo1 "Valor1"))

(Clase2 (Atributo2 "Valor2") (ReferenciasClase1 "Clase1\_1" "Clase1\_2")))

###### **3.6.3.3.3 Resultado Java**

class Clase1 {

private Tipo1 atributo1;

public Clase1(Tipo1 atributo1) {

this.atributo1 = atributo1;

}

public Tipo1 met1() {

return atributo1;

}

}

class Clase2 {

private Tipo2 atributo2;

private List<Clase1> referenciasClase1; // Asociación unidireccional (Clase2 -> Clase1)

public Clase2(Tipo2 atributo2, List<Clase1> referenciasClase1) {

this.atributo2 = atributo2;

this.referenciasClase1 = referenciasClase1;

}

public Tipo2 met2() {

return atributo2;

}

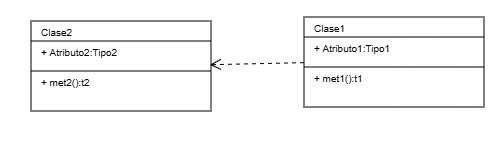
public List<Clase1> getReferenciasClase1() {

return referenciasClase1;

}

}

##### **3.6.3.4 Asociación Dependencia**



###### **3.6.3.4.1 Resultado XML**

<Clases>

<Clase nombre="Clase1">

<Atributo nombre="Atributo1" tipo="Tipo1"/>

<Metodo nombre="met1" tipo="Tipo1"/>

</Clase>

<Clase nombre="Clase2">

<Atributo nombre="Atributo2" tipo="Tipo2"/>

<Metodo nombre="met2" tipo="Tipo2"/>

<Dependencia clase="Clase1"/>

</Clase>

</Clases>

###### **3.6.3.4.2 Resultado Clips**

(deftemplate Clase1

(slot atributo1))

(deftemplate Clase2

(slot atributo2))

(defrule usar-Clase1-en-Clase2

?c1 <- (Clase1 (atributo1 ?a1))

=>

(printout t "Usando Clase1 en Clase2 con atributo: " ?a1 crlf))

###### **3.6.3.4.3 Resultado Java**

class Clase1 {

public Tipo1 atributo1;

public Tipo1 met1() {

return atributo1;

}

}

class Clase2 {

public Tipo2 atributo2;

public Tipo2 met2() {

return atributo2;

}

// Dependencia con Clase1

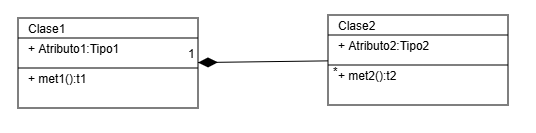
public void usarClase1(Clase1 obj) {

System.out.println("Usando Clase1 en Clase2: " + obj.met1());

}

}

##### **3.6.3.5 Asociación Composición**



###### **3.6.3.5.1 Resultado XML**

<?xml version="1.0" encoding="UTF-8"?>

<UMLModel>

<Class name="Clase1">

<Attribute name="Atributo1" type="Tipo1"/>

<Method name="met1" returnType="Tipo1"/>

</Class>

<Class name="Clase2">

<Attribute name="Atributo2" type="Tipo2"/>

<Method name="met2" returnType="Tipo2"/>

</Class>

<Composition>

<End class="Clase1" multiplicity="1"/>

<End class="Clase2" multiplicity="\*"/>

</Composition>

</UMLModel>

###### **3.6.3.5.2 Resultado Clips**

(deftemplate Clase1

(slot Atributo1)

(multislot InstanciasClase2)) ;; Composición: Contiene múltiples Clase2

(deftemplate Clase2

(slot Atributo2))

(deffunction met1 (?instanciaClase1)

(bind ?valor (fact-slot-value ?instanciaClase1 Atributo1))

(return ?valor))

(deffunction met2 (?instanciaClase2)

(bind ?valor (fact-slot-value ?instanciaClase2 Atributo2))

(return ?valor))

(deffacts Instancias

(Clase1 (Atributo1 "Valor1") (InstanciasClase2 "Clase2\_1" "Clase2\_2"))

(Clase2 (Atributo2 "Valor2"))

(Clase2 (Atributo2 "Valor3")))

###### **3.6.3.5.3 Resultado Java**

class Clase1 {

private Tipo1 atributo1;

private List<Clase2> listaClase2; // Composición: Clase1 contiene instancias de Clase2

public Clase1(Tipo1 atributo1) {

this.atributo1 = atributo1;

this.listaClase2 = new ArrayList<>();

}

public void agregarClase2(Tipo2 atributo2) {

Clase2 nuevaClase2 = new Clase2(atributo2);

listaClase2.add(nuevaClase2);

}

public Tipo1 met1() {

return atributo1;

}

}

class Clase2 {

private Tipo2 atributo2;

public Clase2(Tipo2 atributo2) {

this.atributo2 = atributo2;

}

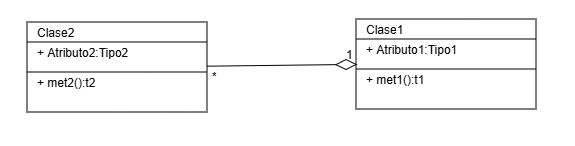
public Tipo2 met2() {

return atributo2;

}

}

##### **3.6.3.6 Asociación Agregación**



###### **3.6.3.6.1 Resultado XML**

<?xml version="1.0" encoding="UTF-8"?>

<UMLModel>

<Class name="Clase1">

<Attribute name="Atributo1" type="Tipo1"/>

<Method name="met1" returnType="Tipo1"/>

</Class>

<Class name="Clase2">

<Attribute name="Atributo2" type="Tipo2"/>

<Method name="met2" returnType="Tipo2"/>

</Class>

<Aggregation>

<End class="Clase1" multiplicity="1"/>

<End class="Clase2" multiplicity="\*"/>

</Aggregation>

</UMLModel>

###### **3.6.3.6.2 Resultado Clips**

(deftemplate Clase1

(slot Atributo1)

(multislot ReferenciasClase2)) ;; Agregación: Referencias a instancias de Clase2

(deftemplate Clase2

(slot Atributo2))

(deffunction met1 (?instanciaClase1)

(bind ?valor (fact-slot-value ?instanciaClase1 Atributo1))

(return ?valor))

(deffunction met2 (?instanciaClase2)

(bind ?valor (fact-slot-value ?instanciaClase2 Atributo2))

(return ?valor))

(deffacts Instancias

(Clase1 (Atributo1 "Valor1") (ReferenciasClase2 "Clase2\_1" "Clase2\_2"))

(Clase2 (Atributo2 "Valor2"))

(Clase2 (Atributo2 "Valor3")))

###### **3.6.3.6.3 Resultado Java**

class Clase1 {

private Tipo1 atributo1;

private List<Clase2> listaClase2; // Agregación: Clase1 tiene referencias a Clase2

public Clase1(Tipo1 atributo1) {

this.atributo1 = atributo1;

this.listaClase2 = new ArrayList<>();

}

public void agregarClase2(Clase2 clase2) {

listaClase2.add(clase2); // Clase1 almacena una referencia a Clase2, pero no la crea

}

public Tipo1 met1() {

return atributo1;

}

}

class Clase2 {

private Tipo2 atributo2;

public Clase2(Tipo2 atributo2) {

this.atributo2 = atributo2;

}

public Tipo2 met2() {

return atributo2;

}

}

## **4. Proyecto en drive**

[](https://drive.google.com/drive/folders/1vbLSwozDTVlQuLO8CzayDGRBtwLh1QGJ)