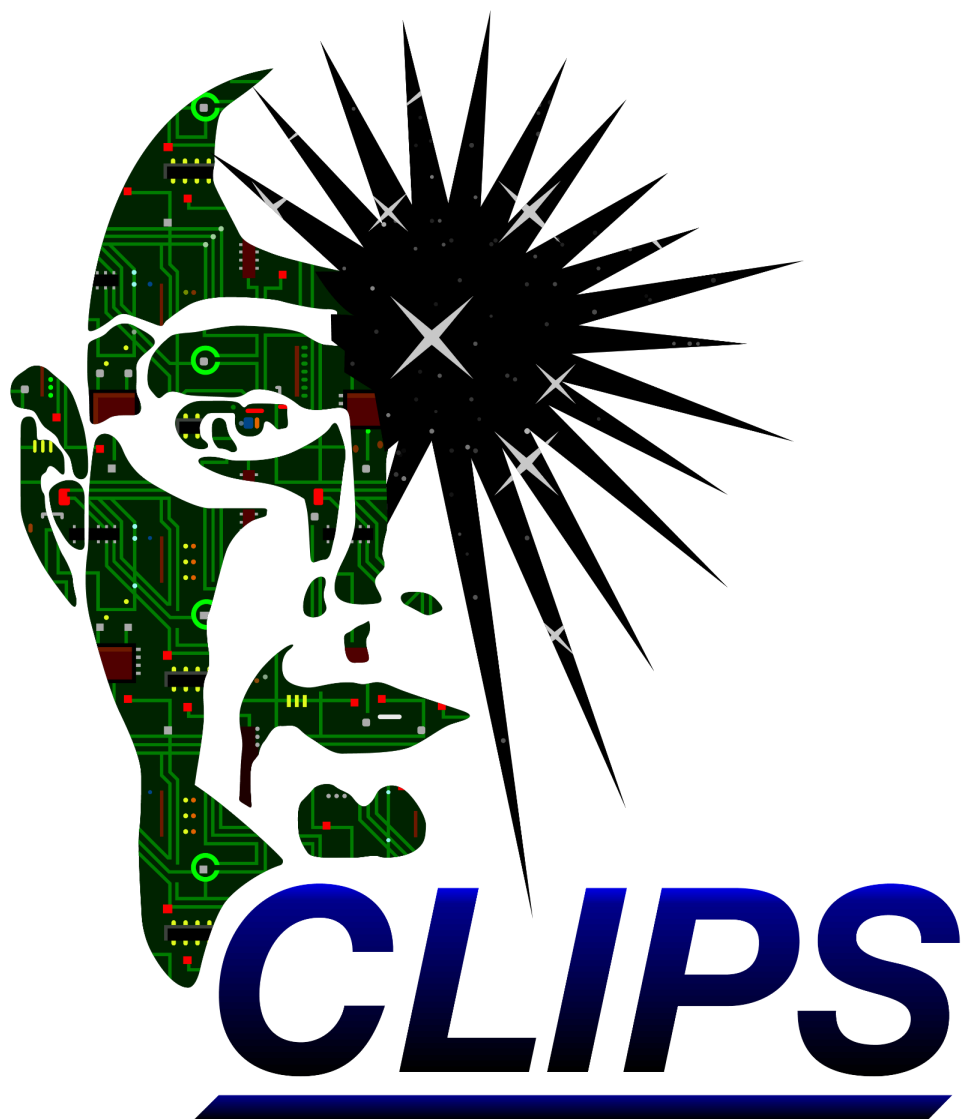


CLIPS - HERRAMIENTA CASE



Adrián Yared Armas de la Nuez

Contenido

1. Apartado 1.....	3
1.1.1 Comando.....	3
2. Apartado 2.....	4
2.1.1 asociación bidireccional.....	4
2.1.1.1 Código.....	5
2.1.1.2 Explicación del código.....	6
2.1.2 composición.....	6
2.1.2.1 Código.....	6
2.1.2.2 Explicación del código.....	7
2.1.3 Agregación.....	7
2.1.3.1 Código.....	7
2.1.3.2 Explicación del código.....	8
3. Apartado 3.....	8
3.1 Actualizar la estructura del proyecto.....	9
3.2 Integrar Flask con Traductor.py.....	9
3.2.1 Comando.....	9
3.3 Configurar el botón en el frontend.....	10
3.3.1 Comando XMI.....	10
3.3.2 Comando Clips.....	10
3.3.3 Comando Java.....	11
3.4 Lógica en JavaScript para la solicitud POST.....	12
3.4.1 Comando.....	12
3.5 Integrar Traductor.py con CLIPS y Clipspy.....	13
2.5.1 Comando.....	13
3.6 Prueba de la integración.....	14
3.6.1 Ejecuta app.py.....	14
3.6.2 Ve a http://127.0.0.1:5000	14
3.6.3 Genera un diagrama UML y presiona el botón para procesarlo.....	14
3.6.3.0.1 Descarga XMI.....	14
3.6.3.0.2 Descarga Clips.....	15
3.6.3.0.3 Descarga Java.....	15
3.6.3.1 Herencia.....	16
3.6.3.1.1 Resultado XML.....	16
3.6.3.1.2 Resultado Clips.....	16
3.6.3.1.2 Resultado Java.....	17
3.6.3.2 Asociación.....	17



CLIPS - HERRAMIENTA CASE

3.6.3.2.1 Resultado XML.....	17
3.6.3.2.2 Resultado Clips.....	18
3.6.3.2.3 Resultado Java.....	18
3.6.3.3 Asociación direccional.....	19
3.6.3.3.1 Resultado XML.....	19
3.6.3.3.2 Resultado Clips.....	19
3.6.3.3.3 Resultado Java.....	20
3.6.3.4 Asociación Dependencia.....	21
3.6.3.4.1 Resultado XML.....	21
3.6.3.4.2 Resultado Clips.....	21
3.6.3.4.3 Resultado Java.....	21
3.6.3.5 Asociación Composición.....	22
3.6.3.5.1 Resultado XML.....	22
3.6.3.5.2 Resultado Clips.....	23
3.6.3.5.3 Resultado Java.....	23
3.6.3.6 Asociación Agregación.....	24
3.6.3.6.1 Resultado XML.....	24
3.6.3.6.2 Resultado Clips.....	24
3.6.3.6.3 Resultado Java.....	25
4. Proyecto en drive.....	26

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 XML. 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

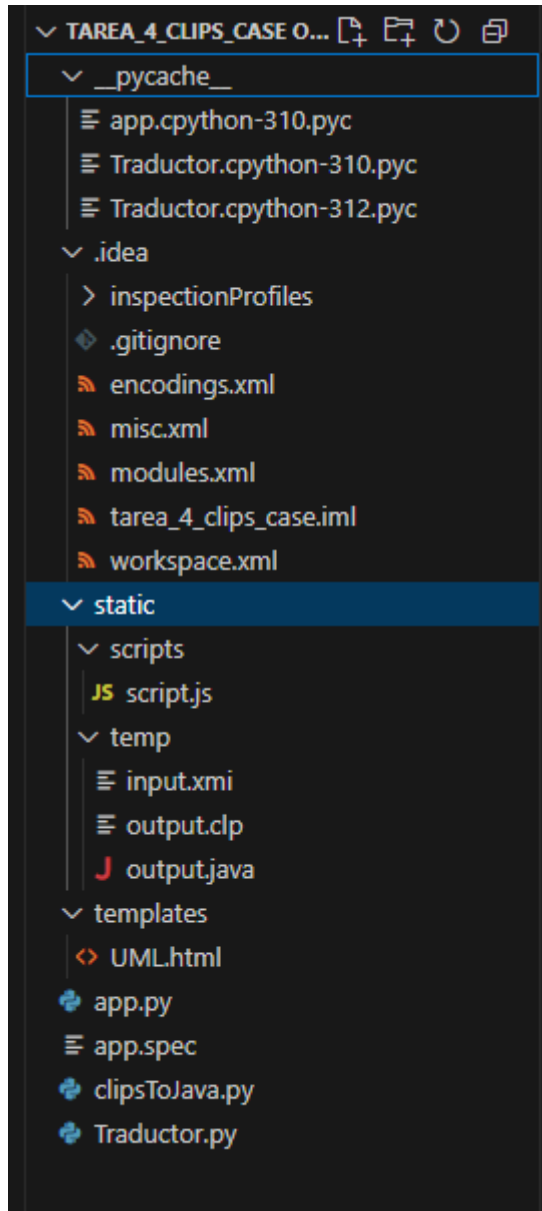
La función es prácticamente idéntica a la de la composición, pero en este caso, las relaciones extraídas son de tipo `uml:Aggregation`. El comportamiento es similar: se extraen las clases `whole` (contenedor) y `part` (contenida), junto con sus multiplicidades. La diferencia clave radica en que, a diferencia 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 Clipsy

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

2.5.1 Comando

Import:

```
from clips import Environment
```

Al final de Traductor.py, se agrega una función que usa clipsy 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

```
PS C:\Users\Adrian\Desktop\tarea_4_clips_case original> & C:/Users/Adrian/anaconda3/python.exe "c:/Users/Adrian/Desktop/tarea_4_clips_case original/app.py"
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

3.6.2 Ve a <http://127.0.0.1:5000>

Al ir a la ruta se ve lo siguiente:

Nombre de la Clase:

Atributo: Visibilidad: Tipo:


Método: Visibilidad: Tipo:

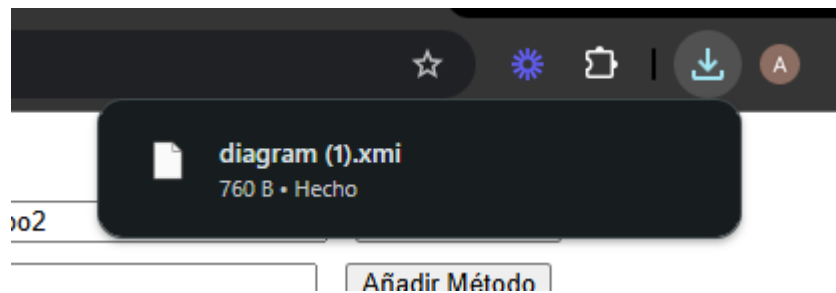
Clase Origen: Clase Destino: Tipo de Relación: Multiplicidad Origen: Multiplicidad Destino:

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

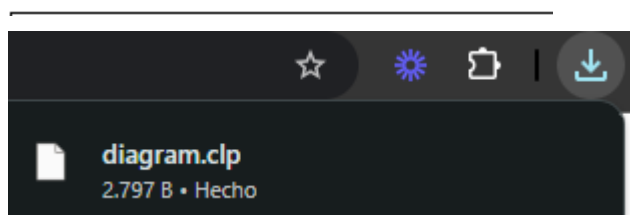
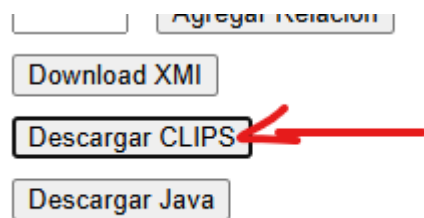
3.6.3.0.1 Descarga XML

Multiplicidad Destino:

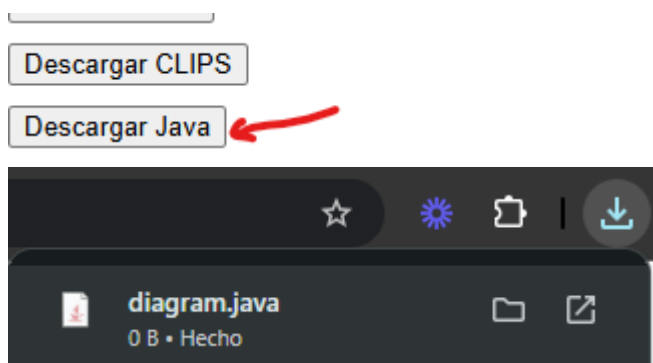




3.6.3.0.2 Descarga Clips



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"?>
  <XML xmi.version="2.1" xmlns:xmi="http://schema.omg.org/spec/XML/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>
  </XML>
  
```

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;
    }
}

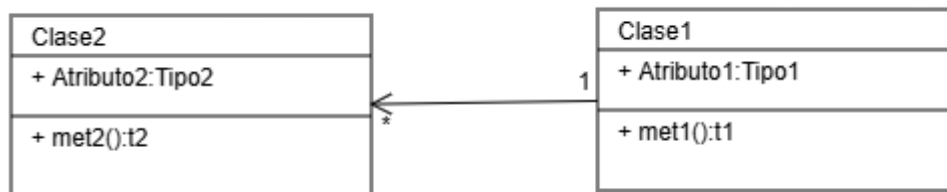
class Clase2 {
    private Tipo2 atributo2;
    private Clase1 clase1; // Asociación con Clase1 (1 a muchos)

    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;
    }
}

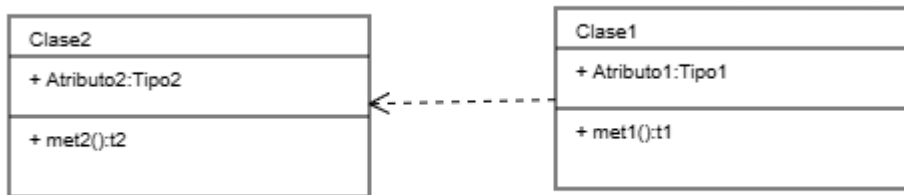
class Clase2 {
    private Tipo2 atributo2;
    private List<Clase1> referenciasClase1; // Asociación unidireccional (Clase2 -> Clase1)

    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;
    }
}

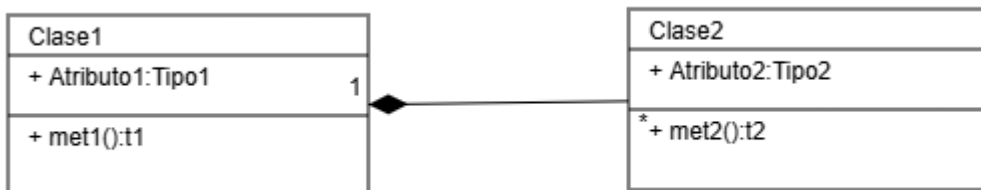
class Clase2 {
    public Tipo2 atributo2;

    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))

(defun met1 (?instanciaClase1)
  (bind ?valor (fact-slot-value ?instanciaClase1 Atributo1))
  (return ?valor))

(defun 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

