UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

INFORME DE PRÁCTICA CALIFICADA 4

TÓPICOS EN CIENCIAS DE LA COMPUTACIÓN

2025

Alumnos

Código	Nombres y Apellidos
u201118258	Alvarez Orellana Iam Anthony Marcelo
u20211d574	Mallma Villanueva Fabiana Nayeli
u202115844	Mancilla Cienfuegos Paula Jimena
u201916314	Pastor Salazar Tomas Alonso

Índice

Descripción del problema

El sistema simula un entorno aeroportuario simplificado con los siguientes roles:

- **Aviones (agentes Avión):** informan que están volando y solicitan aterrizar. Si se les niega el aterrizaje, esperan un tiempo y reintentan.
- **Torre de control (agente Torre):** recibe las solicitudes y autoriza el aterrizaje solo si la pista está libre.
- **Pista:** recurso compartido entre todos los aviones, puede estar libre u ocupada.
- **Visualizador:** representa gráficamente el estado de los agentes.

Esta simulación replica la lógica de sistemas multiagente donde las entidades toman decisiones en tiempo real basadas en su percepción local. Es un ejercicio práctico para aplicar el paradigma de agentes autónomos y comunicación asincrónica, útil en robótica, sistemas distribuidos y simulaciones sociales.

Configuración del entorno

La configuración detallada está documentada en la guía adjunta (**Configurar PC4.pdf**). Los pasos principales fueron:

Entorno base

- Sistema: **Windows + WSL (Ubuntu)**
- Python: 3.10.18 (usando conda)
- SPADE: instalación vía *pip install spade*
- Prosody: servidor XMPP para la comunicación entre agentes

Esta combinación permite separar la interfaz gráfica (en Windows) del backend de agentes (en Linux), fomentando el trabajo con entornos heterogéneos como ocurre en sistemas reales.

Pasos clave

1. Crear entorno conda:

```
conda create -n pc4-topicos1
conda activate pc4-topicos1
```

2. Instalar SPADE:

```
pip install spade
```

3. Configurar Prosody:

```
sudo apt update
sudo apt install prosody
sudo vim /etc/prosody/prosody.cfg.lua
```

Agregar:

```
c2s_require_encryption = false
allow_unencrypted_plain_auth = true

VirtualHost "localhost"
    enabled = true
    ssl = {
        key = "/dev/null";
        certificate = "/dev/null";
    }
```

4. Reiniciar y registrar agentes:

```
sudo systemctl restart prosody

sudo prosodyctl register torre localhost 1234
sudo prosodyctl register avion0 localhost 1234
sudo prosodyctl register avion1 localhost 1234
sudo prosodyctl register avion2 localhost 1234
```

Este paso fue crucial para permitir que cada agente tenga una identidad XMPP válida con la que pueda enviar y recibir mensajes.

Implementación de los agentes

La estructura del proyecto en Visual Studio Code es la siguiente:

```
PC4-TOPICOS/
│
├── agentes/
│   ├── avion_agent.py
│   └── torre_agent.py
├── visual/
│   └── visualizador.py
├── shared_state.py
└── main.py
```

Agente Avión (*avion_agent.py*)

- Hereda de *spade.agent.Agent* .
- Tiene un comportamiento cíclico que:
    - Vuela durante un tiempo aleatorio.
    - Solicita aterrizaje a la torre.
    - Espera respuesta y actúa según el permiso.
    - Cambia de estado global para la visualización.

El comportamiento del avión emula un sistema autónomo reactivo, capaz de esperar y reintentar en caso de negativa. Utiliza *asyncio* para mantener concurrencia no bloqueante.

Agente Torre (*torre_agent.py*)

- Controla la pista (libre/ocupada).
- Responde a los mensajes de los aviones.
- Da permiso si la pista está libre y la bloquea por 5 segundos.

Este agente representa un componente centralizado con recursos limitados, similar a un semáforo de tráfico aéreo.

Visualización con Pygame

El archivo *visualizador.py* utiliza **Pygame** para representar visualmente:

- El estado de cada avión (volando, solicitando, esperando, aterrizando).
- La torre ('t') y la pista ('p').
- La trayectoria de los aviones según su estado.
- Los aviones que aterrizan desaparecen del sistema.

Además, se usan identificadores simples como *a0* , *a1* , *a2* , lo cual facilita el rastreo en pantalla sin sobrecargar la interfaz. Esta visualización se actualiza a partir de un estado compartido sincronizado con los agentes.

La implementación de *pygame* fue útil para observar en tiempo real el comportamiento y evolución del sistema multiagente. El uso de *multiprocessing* permite que la interfaz visual y la lógica del sistema trabajen en paralelo sin interferencias.

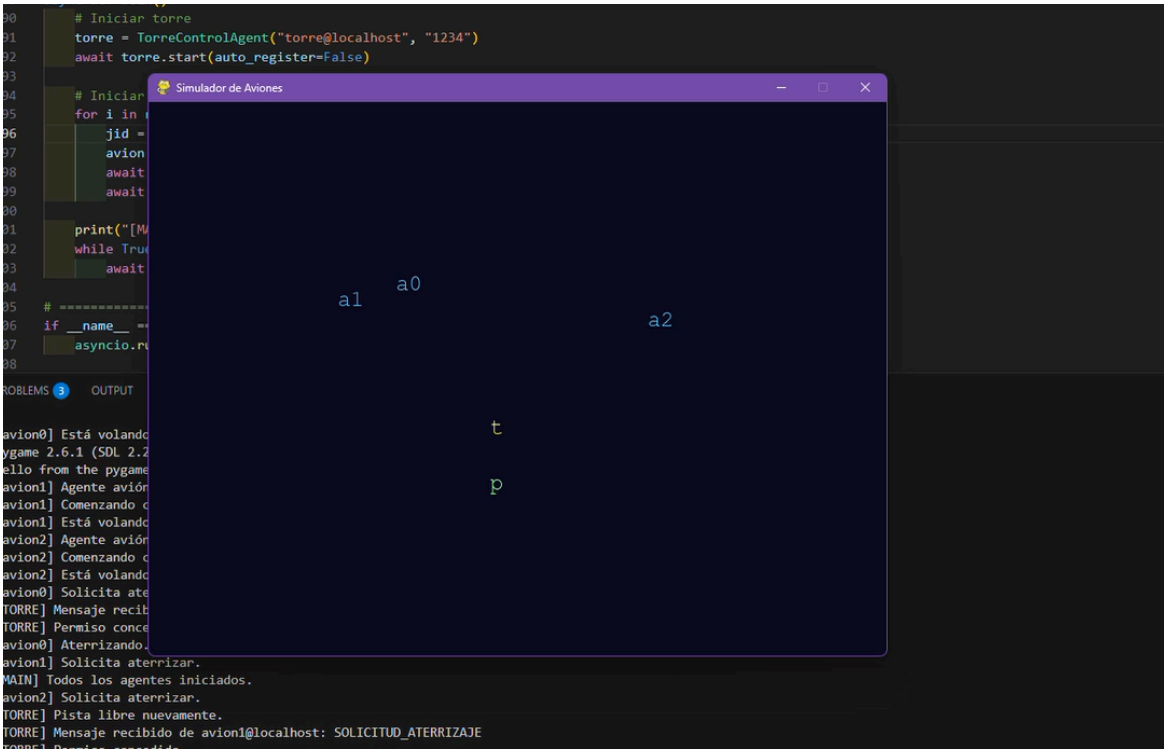
Ejecución general del sistema

El archivo *main.py* :

- Lanza el proceso de visualización en paralelo con *multiprocessing.Process* .
- Inicia la torre y tres agentes avión con intervalos.
- Establece un estado compartido (*Manager().dict*) para sincronizar la visualización.

Luego, se lanza un bucle asincrónico que puede crear nuevos agentes avión cada cierto tiempo, lo que permite simular un flujo dinámico e ilimitado de aviones si ya están registrados previamente.

Cada componente funciona de manera concurrente y sincronizada a través del sistema de mensajes y el diccionario compartido. Esto demuestra una arquitectura distribuida coordinada mediante comunicación por mensajes XMPP.



Conclusiones

Ventajas de SPADE frente a JADE:

- ✓ SPADE es más ligero y rápido de configurar.
- ✓ Uso directo de Python facilita la integración con otras librerías (como Pygame).
- ✓ Permite usar procesos nativos y asincronía con *asyncio* .

Desventajas:

- ✗ Comunidad más pequeña que JADE.
- ✗ Documentación limitada para casos avanzados.
- ✗ Requiere configuración adicional (como Prosody) en WSL para funcionar en Windows.

Reflexión final

El uso de SPADE permitió modelar agentes inteligentes que actúan de forma autónoma y cooperan entre sí sin control centralizado. La visualización ayudó a comprender mejor las interacciones y estados del sistema.

La práctica también sirvió para consolidar conocimientos sobre asincronía, procesos concurrentes, comunicación en sistemas distribuidos y representación visual de información dinámica.

Uso de IA generativa (GPT)

Para el desarrollo de esta práctica, se hizo uso de herramientas de inteligencia artificial (como ChatGPT) en los siguientes aspectos del trabajo:

- **Estructura del informe:** Se utilizó GPT para organizar las secciones del informe, redactar introducciones y mantener un estilo claro, conciso y técnico.

- **Estructura del proyecto:** Se solicitó apoyo para estructurar correctamente el proyecto en carpetas, asignar responsabilidades a cada archivo y validar el flujo de agentes.
- **Revisión ortográfica:** Se empleó GPT para corregir errores gramaticales y mejorar la redacción general del documento.

La asistencia de IA permitió enfocarnos más en la implementación del código y en el análisis del comportamiento del sistema.

Bibliografía

- SPADE documentation: https://spade-mas.readthedocs.io/
- Prosody XMPP Server: https://prosody.im/
- Pygame Docs: https://www.pygame.org/docs/

Anexo

- **Enlace al repositorio en Gitlab:** https://gitlab.com/groppoxx-group/pc4-project.git
- **Enlace al repositorio en GitHub:** https://github.com/Groppoxx/topicos-pc4-project.git
- **Imagen del proyecto en Visual Studio Code:**