Comparación de JADE y SPADE para la generación de horarios académicos mediante sistemas multiagentes

Comparison of JADE and SPADE for academic timetable generation using multi-agent systems

Gabriel Icarte-Ahumada[1*] https://orcid.org/0000-0002-1997-0053
Franco Calderón Meza[1] https://orcid.org/0009-0009-7855-9633

Recibido 25 de agosto de 2025, aceptado 23 de diciembre de 2025
Received: August 25, 2025 *Accepted: December 23, 2025*

RESUMEN

La generación de horarios académicos es un tipo de problema de scheduling que requiere la asignación eficiente de recursos bajo múltiples restricciones. Tradicionalmente, este tipo de problemas ha sido abordado mediante métodos centralizados, heurísticos y metaheurísticos, que si bien ofrecen soluciones válidas, presentan limitaciones frente a la escalabilidad y adaptabilidad. En este contexto, los sistemas multiagentes (MAS) han demostrado ser una alternativa efectiva para resolver problemas de scheduling, gracias a su enfoque distribuido, su capacidad de negociación entre agentes y su adaptabilidad a entornos dinámicos. Para desarrollar sistemas multiagentes, existen diversas plataformas las cuales difieren en aspectos como el lenguaje de programación, el modelo de concurrencia, el soporte de estándares como FIPA, la facilidad de uso, el rendimiento técnico y la adecuación a distintos escenarios de implementación. Este artículo presenta un estudio comparativo entre dos plataformas multiagente, JADE y SPADE, para resolver un problema de scheduling. Utilizando un caso de estudio en la Universidad Arturo Prat, se diseñó un sistema basado en el protocolo Contract Net para coordinar la asignación de profesores, asignaturas y salas. Se evaluaron las plataformas según métricas de calidad de los horarios obtenidos y el uso de los recursos computacionales. Los resultados muestran que JADE ofrece mayor estabilidad y eficiencia bajo alta carga, mientras que SPADE presenta ventajas en escenarios de baja complejidad.

Palabras clave: Sistemas multiagentes, scheduling, timetabling, JADE, SPADE.

ABSTRACT

Academic timetabling is a scheduling problem that involves efficiently allocating resources while satisfying multiple constraints. Traditionally, these problems have been addressed through centralized, heuristic, and metaheuristic approaches. Although these methods can produce feasible solutions, they often exhibit limitations in scalability and adaptability. In this context, multi-agent systems (MAS) have emerged as an effective alternative for solving scheduling problems, thanks to their distributed architecture, agent negotiation capabilities, and adaptability to dynamic environments. Various platforms are available for developing MAS, differing in programming languages, concurrency models, support for standards such as FIPA, usability, technical performance, and suitability for different deployment scenarios. This article presents a comparative study of two multi-agent platforms, JADE and SPADE, as applied to a scheduling

[1] Universidad Arturo Prat. Facultad de Ingeniería y Arquitectura. Iquique, Chile. E-mail: gicarte@unap.cl; fcalderonm@estudiantesunap.cl
* Autor de correspondencia: gicarte@unap.cl

DOI: 10.64966/ingeniare.v33.31.

problem. A case study at Universidad Arturo Prat involved developing a system based on the Contract Net Protocol to coordinate the assignment of instructors, courses, and classrooms. The platforms were evaluated using metrics that capture both the quality of the resulting timetables and the computational resource consumption. The results indicate that JADE offers greater stability and efficiency under high-load conditions, while SPADE displays advantages in less complex scenarios.

Keywords: Multi-agent systems, scheduling, timetabling, JADE, SPADE.

INTRODUCCIÓN

Los problemas de scheduling consisten en asignar recursos limitados, tales como el tiempo, personal, equipos o espacios, a un conjunto de tareas que deben cumplirse bajo restricciones diversas, optimizando criterios como makespan, utilización de recursos o nivel de servicio [1]. Se trata de problemas NP-duros, cuya complejidad crece exponencialmente con el tamaño de la instancia. Un tipo de problema de scheduling es el timetabling educativo, donde las actividades académicas deben ubicarse en bloques y salas disponibles respetando capacidad, secuenciación y preferencias de docentes y estudiantes [2], [3].

Para abordarlos, la literatura ha propuesto métodos centralizados basados en programación entera, así como heurísticas y metaheurísticas (recocido simulado, búsqueda tabú, algoritmos genéticos). Estas técnicas logran soluciones aceptables en plazos razonables y con implementaciones relativamente sencillas. Sin embargo, adolecen de falta de escalabilidad, poca adaptabilidad a cambios en línea y dependencia de un punto único de fallo. Su rendimiento suele degradarse cuando se incrementa el número de restricciones o cuando las preferencias cambian dinámicamente.

Una alternativa viable para superar estas limitaciones son los sistemas multiagentes. Un sistema multiagente es un conjunto de agentes autónomos que interactúan entre sí y con su entorno para resolver problemas complejos de manera distribuida, permitiendo la coordinación, cooperación y negociación entre ellos [4]. En el ámbito del scheduling, incluido el timetabling, han demostrado mejoras en adaptabilidad y tiempos de respuesta, especialmente cuando se combinan con protocolos de negociación como Contract Net. No obstante, al igual que las heurísticas y metaheurísticas, los sistemas multiagentes no garantizan en general la obtención del óptimo global del problema, ya que la calidad de las soluciones depende del diseño de los agentes, de las reglas de negociación y de los parámetros de configuración. En base a lo anterior, el presente trabajo se plantea como un estudio exploratorio que ilustra el potencial de este enfoque en un caso real.

Para el desarrollo de sistema multiagentes hay varias alternativas. Entre las más citadas se encuentran JADE, escrito en Java y alineado con los estándares FIPA; SPADE, basado en Python con arquitectura asíncrona sobre XMPP; MadKit y Jason, entre otras [5], [6]. Cada una ofrece distintos niveles de soporte de herramientas, rendimiento y facilidad de integración, por lo que es importante conocer y comparar su idoneidad para problemas de scheduling reales.

Este artículo tiene como propósito presentar una comparación entre dos plataformas de desarrollo de sistemas multiagente orientadas a la resolución de problemas de scheduling. En ambas plataformas se implementó un sistema de generación de horarios académicos, con idéntica estructura (se mantienen los mismos tipos de agentes, roles, comportamientos, protocolo de interacción, flujo de mensajes y parámetros del escenario experimental). En el artículo se describen la metodología empleada, los escenarios simulados, las métricas de evaluación consideradas, los resultados obtenidos y su discusión comparativa, con el fin de contribuir al análisis de fortalezas y debilidades de estas plataformas en contextos reales.

TRABAJOS RELACIONADOS

La generación de horarios, o timetabling, se refiere al proceso de asignar actividades, recursos y bloques de tiempo de manera óptima, y ha sido ampliamente estudiada debido a su complejidad y a su amplia gama de aplicaciones prácticas en contextos educativos, industriales y de transporte. En el caso particular del timetabling académico, un punto de partida

es el trabajo realizada por Ceschia [2], el cual proporciona una taxonomía detallada de variantes del problema, instancias utilizadas en competencias internacionales y enfoques algorítmicos aplicados en los últimos años. En [7] proponen un enfoque híbrido que combina heurísticas de ordenamiento con hipermetaheurísticas para abordar la asignación de cursos. En [8] introducen una clasificación de instancias basada en características del problema, facilitando la selección de algoritmos adecuados según el tipo de instancia. En [9] muestran una solución que optimiza la utilización de salas en la Universidad de Lisboa mediante modelos combinatorios, evidenciando la aplicabilidad práctica de estas técnicas en contextos reales.

Diversos estudios han explorado el uso de sistemas multiagentes (MAS) para abordar el problema de timetabling en instituciones educativas, destacando sus ventajas en términos de descentralización, adaptabilidad y negociación distribuida. En [10] desarrollaron un sistema basado en la plataforma Prometheus, compuesto por agentes que representan cursos, profesores y recursos, mostrando que un enfoque colaborativo mejora la eficiencia de los horarios generados. En [11] propusieron un MAS integrado a un sistema de apoyo a decisiones que considera las preferencias de los usuarios, logrando una generación automática de horarios más personalizada. En [12] presentan MAS-UP-UCT, una arquitectura de agentes que minimiza la necesidad de comunicación excesiva, logrando una asignación eficiente mediante negociación distribuida. En [13] implementaron un sistema de agentes especializados para resolver el problema de horarios, destacando la flexibilidad del enfoque. En [14] propusieron un modelo iterativo en el cual los agentes eliminan propuestas no preferidas hasta alcanzar una solución aceptable para todos los participantes.

La investigación sobre plataformas para el desarrollo de sistemas multiagentes ha dado lugar a múltiples estudios comparativos y descriptivos que orientan la elección tecnológica según las necesidades de un problema. En [6] ofrecen una revisión de plataformas MAS, clasificándolas en herramientas activas, comerciales e históricas, y proponiendo criterios técnicos para evaluar su idoneidad. En [15] presentan SPADE 3, una evolución significativa de esta plataforma basada en Python, que incorpora mejoras en interoperabilidad, soporte para el modelo BDI y una arquitectura asíncrona optimizada. En [16], introducen UMAP, una plataforma MAS desarrollada en .NET, destacada por su compatibilidad con FIPA y su orientación a entornos empresariales y en la nube. En [17] muestran una comparación entre JADE y SPADE. Analizan su rendimiento, escalabilidad y facilidad de uso, proporcionando evidencia empírica sobre sus ventajas y limitaciones. En [18] comparan JADE, SPADE y Jadex, considerando tanto su popularidad académica como su usabilidad en aplicaciones industriales, destacando las diferencias en GUI, soporte técnico e interoperabilidad. Estos trabajos aportan una visión integral sobre las fortalezas, debilidades y casos de uso más apropiados para cada plataforma. Para el conocimiento de los autores, no existen comparaciones entre plataformas para el desarrollo de sistemas multiagentes utilizando un problema de timetabling como escenario común.

A partir del análisis de los trabajos descritos previamente, la revisión de la literatura muestra brechas en el estudio de sistemas multiagente aplicados al scheduling de actividades académicas y, en general, a contextos de asignación de recursos con restricciones complejas. En los estudios revisados no se identifican comparaciones de rendimiento y calidad de las soluciones entre plataformas de desarrollo MAS que consideren simultáneamente criterios de rendimiento, escalabilidad y facilidad de modelado, lo que refuerza la pertinencia del enfoque propuesto en este artículo. Los estudios que sí comparan plataformas MAS como JADE, SPADE o MADKIT suelen enfocarse en criterios generales de rendimiento o arquitectura, pero no utilizan problemas de scheduling como base experimental común. Esto dificulta extraer conclusiones prácticas sobre cuál plataforma resulta más adecuada en contextos de asignación de recursos con restricciones complejas.

El presente artículo busca abordar estas brechas mediante una evaluación comparativa entre dos plataformas multiagentes, JADE y SPADE, aplicadas a un mismo problema de timetabling académico. A diferencia de trabajos previos, se implementa una arquitectura MAS idéntica en ambas plataformas, lo que permite una comparación justa basada en métricas tanto de calidad de solución como de

rendimiento técnico. El estudio incorpora escenarios experimentales de diferente escala, lo que permite analizar el comportamiento de cada plataforma bajo distintas condiciones de carga.

METODOLOGÍA

La metodología se desarrolló en tres fases. La primera corresponde a la identificación del problema y las restricciones. En la segunda se diseñó el sistema multiagente, definiendo los roles y comportamientos de los agentes y el protocolo de interacción. La tercera fase estableció las métricas académicas y técnicas empleadas para evaluar la calidad de los horarios y el rendimiento computacional. La Figura 1 muestra los aspectos más relevantes de la metodología.

Identificación del problema

El problema de timetabling abordado en este estudio corresponde a la generación automatizada de horarios académicos para la Facultad de Ingeniería y Arquitectura de la Universidad Arturo Prat, bajo un conjunto específico de restricciones operativas y pedagógicas. Estas restricciones buscan mejorar la calidad de los horarios mediante un uso eficiente de los espacios físicos, la adecuación a las necesidades de los académics y la reducción de conflictos de asignación.

Las siguientes ocho directrices fueron consideradas en la implementación del sistema:

1. Las actividades curriculares deben asignarse entre las 8:00 y las 18:30 horas, lo que equivale a un total de 9 bloques académicos por día, siendo cada bloque una hora cronológica.
2. El horario de una actividad curricular no puede exceder más de dos bloques continuos, salvo en el caso de talleres y laboratorios.
3. Los estudiantes de primer año (semestres 1 y 2) deben mantener prioritariamente clases en la jornada de la mañana.
4. Las actividades deben concentrarse preferentemente en el campus correspondiente a la carrera, con excepción de actividades de formación general o ciencias básicas.
5. Cuando una cohorte tiene clases en más de un campus, debe existir al menos un bloque de separación entre dichas actividades, y no debe ocurrir más de un traslado por día.
6. Se debe evitar traslados innecesarios entre campus, considerando que estos se encuentran distantes entre sí y que los estudiantes requieren transporte público para desplazarse.
7. Cada cohorte debe tener un horario intercalado: los estudiantes de primer, tercer y quinto año preferentemente en la mañana; y los de segundo, cuarto y sexto año en la tarde.
8. Las actividades curriculares que consideren entre 9 y 70 estudiantes deben ser asignadas a salas de clases regulares. Si el número de estudiantes es inferior a 9, se deben usar salas de reuniones; si excede los 70, se deben generar paralelos equivalentes.

Figura 1. Metodología utilizada.

Diseño del sistema

El sistema multiagente (MAS) desarrollado tiene como objetivo resolver el problema de timetabling académico mediante la asignación automatizada de horarios, cursos y salas, respetando un conjunto definido de restricciones institucionales. El sistema está diseñado para operar de forma descentralizada, permitiendo que agentes autónomos, cada uno con un rol específico, interactúen y negocien asignaciones de forma colaborativa. Esta sección detalla la estructura funcional del sistema, incluyendo la definición y comportamiento de los agentes, el protocolo de interacción utilizado para la negociación y el mecanismo de toma de decisiones adoptado por cada tipo de agente.

Agentes

El sistema contempla tres tipos principales de agentes: agentes profesor, agentes sala y un agente supervisor. Cada uno cumple funciones específicas dentro del proceso de planificación y opera de manera autónoma bajo un modelo distribuido de coordinación.

El agente profesor representa a un docente que debe dictar una o más asignaturas. Su principal función es iniciar el proceso de asignación horaria solicitando propuestas a los agentes sala. Este agente analiza las respuestas recibidas y selecciona las que mejor se ajustan a las restricciones curriculares, como la capacidad de la sala, la continuidad horaria, la ubicación del campus o la preferencia por horarios matutinos o vespertinos. Una vez finalizado su proceso de asignación, transfiere el control al siguiente agente profesor, siguiendo un modelo secuencial de turnos.

El agente sala representa un espacio físico disponible para impartir clases. Responde a las solicitudes enviadas por los agentes profesor, generando propuestas de uso horario en función de su disponibilidad, capacidad y restricciones físicas. El agente sala puede rechazar solicitudes si no tiene bloques disponibles o si la asignatura no se ajusta a sus condiciones. Una vez aceptada su propuesta por parte del profesor, actualiza su calendario interno y queda a la espera de nuevas interacciones.

El agente supervisor tiene una función de control y cierre del sistema. Su rol principal es monitorear el proceso global de asignación y emitir una señal de terminación una vez que todos los agentes profesor han finalizado sus negociaciones. También asegura el almacenamiento correcto de los resultados en archivos estructurados para su posterior visualización.

Protocolo de interacción

La interacción entre agentes en el sistema se basa en el protocolo Contract Net Protocol (CNP) [19], ampliamente utilizado en sistemas multiagente para la asignación distribuida de tareas. El CNP sigue una lógica de licitación, en la cual un agente actúa como solicitante (manager) y otros como ofertantes (contractors). En este sistema, los agentes profesor asumen el rol de solicitantes y los agentes sala el de ofertantes.

El proceso se inicia cuando un agente profesor envía un Call for Proposals (CFP) a todos los agentes sala, especificando los requisitos de la asignatura que desea asignar, tales como el número de vacantes, tipo de actividad y campus preferido. Los agentes sala evalúan internamente si pueden cumplir con la solicitud y, en caso afirmativo, envían una propuesta con la información del horario disponible, la sala y un puntaje estimado de adecuación. Si no pueden satisfacer los requisitos, responden con un mensaje de rechazo (REFUSE). El agente profesor espera un conjunto de propuestas, evalúa su idoneidad y responde seleccionando la mejor opción mediante un mensaje de aceptación (ACCEPT_PROPOSAL) y descartando las otras propuestas con un REJECT_PROPOSAL. Este protocolo asegura flexibilidad, escalabilidad y un mecanismo distribuido de toma de decisiones sin requerir control centralizado. La Figura 2 muestra el protocolo.

Toma de decisiones de los agentes

El agente profesor debe decidir cuál de las propuestas recibidas acepta para asignar una actividad académica. Para ello, evalúa cada propuesta mediante una fórmula que integra bonificaciones y penalizaciones asociadas a criterios institucionales y preferencias docentes. En el sistema desarrollado, las bonificaciones consideradas incluyen:

- Coincidencia del campus entre actividad y sala (+1),
- Asignación de bloques continuos (+1),
- Preferencia del profesor por bloques horarios determinados (+1),

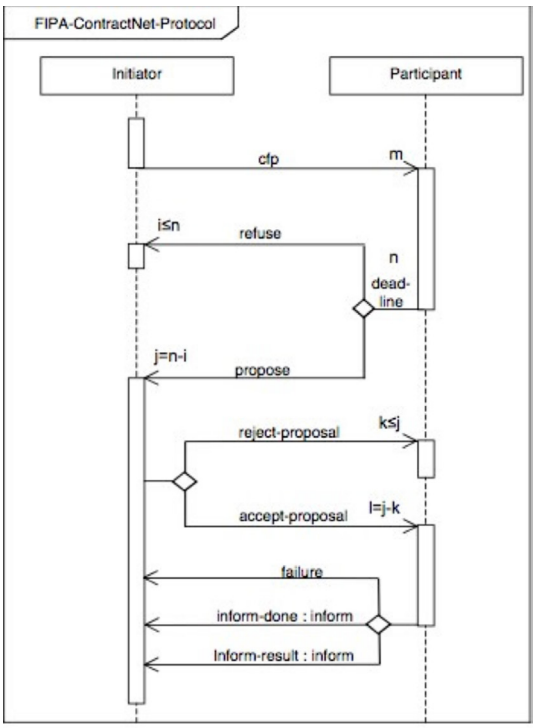

Figura 2. Contract-net protocol.

Por otro lado, las penalizaciones que reducen el puntaje de una propuesta son:

- Diferencia de campus entre actividad y sala (–1),
- Asignar más de una actividad a un mismo profesor en un mismo día (–1),
- Aumento de la cantidad de días con clases para un mismo profesor (–1 por día adicional),
- Duplicidad de asignaturas el mismo día (–1 por conflicto detectado).

Una vez evaluadas todas las propuestas recibidas, el agente selecciona la que posee el mayor puntaje.

El agente sala evalúa internamente si puede satisfacer una solicitud del agente profesor. Dado un conjunto de bloques disponibles, una solicitud es aceptada si existe una secuencia continua de bloques disponibles. Si se cumple esta condición, el agente genera una propuesta con la información del bloque, sala y grado de adecuación.

El agente supervisor no toma decisiones sobre asignaciones, pero sí decide cuándo finalizar la ejecución. Define una condición de parada basada en el número de agentes profesor que han finalizado. Cuando esta condición se cumple, el supervisor emite un mensaje de finalización global al sistema.

Implementación

Para garantizar la comparabilidad se implementó idéntica lógica de negocio en dos frameworks. JADE y SPADE.

JADE (Java Agent DEvelopment Framework) [20] es una plataforma de desarrollo de sistemas multiagentes completamente implementada en Java que permite crear, gestionar y desplegar agentes distribuidos de manera eficiente. Está diseñada conforme a las especificaciones de la Foundation for Intelligent Physical Agents (FIPA), lo que garantiza interoperabilidad y estandarización en la comunicación entre agentes. JADE proporciona un entorno robusto que incluye servicios de descubrimiento (Directory Facilitator), gestión de agentes (Agent Management System), y una interfaz gráfica que facilita la supervisión y control del sistema en tiempo real. Gracias a su arquitectura basada en contenedores y su compatibilidad con múltiples sistemas operativos, JADE permite distribuir agentes en diferentes nodos de red, favoreciendo la escalabilidad. Esta plataforma es ampliamente utilizada en el ámbito académico e industrial debido a su madurez, extensa documentación, comunidad activa y soporte para aplicaciones móviles y distribuidas.

SPADE (Smart Python Agent Development Environment) [15] es una plataforma para el desarrollo de sistemas multiagentes basada en el lenguaje Python, orientada a facilitar la creación de agentes autónomos que se comunican utilizando el protocolo XMPP. Su arquitectura aprovecha el modelo de programación asíncrona de Python (async/await), lo que permite gestionar múltiples comportamientos concurrentes de forma eficiente sin necesidad de hilos tradicionales. SPADE proporciona componentes como servicios de mensajería, localización de agentes y mecanismos de persistencia, y aunque no está completamente alineada con el estándar FIPA, incorpora conceptos clave como el uso de ACLs (Agent Communication Language) y comportamientos estructurados. Una de sus fortalezas radica en la simplicidad de uso y su integración con tecnologías modernas, lo que la

convierte en una opción atractiva para prototipos rápidos, sistemas distribuidos ligeros y aplicaciones que requieren comunicación en red en tiempo real. Si bien su comunidad es más reducida que la de JADE, cuenta con documentación clara y una base modular que facilita la extensión y personalización del sistema.

Métricas de evaluación

Para realizar una comparación entre las dos plataformas, se establecieron un conjunto de métricas de evaluación. Estas métricas se agrupan en dos categorías: i) métricas asociadas al uso de recursos computacionales y ii) métricas que evalúan la calidad de la solución generada.

Métricas de recursos computacionales

Estas métricas cuantifican el consumo de recursos del sistema durante la ejecución del algoritmo multiagente, permitiendo comparar la eficiencia de las plataformas en términos de carga computacional y escalabilidad [21].

- Uso de CPU (%): Mide el porcentaje de utilización del procesador durante el ciclo completo de ejecución del sistema multiagente.
- Uso de Memoria (MB): Corresponde a la cantidad de memoria RAM utilizada por la plataforma MAS y los agentes en ejecución.
- Tiempo de Round Trip (RTT): Se define como el tiempo promedio que tarda un mensaje en ir desde un agente emisor a un agente receptor y retornar con una respuesta.

Métricas de calidad de la solución

Estas métricas evalúan qué tan bien la solución generada por el sistema cumple con los objetivos funcionales del problema de *timetabling*, considerando restricciones duras y blanda [2].

- Tasa de Ocupación de Salas (%): Calcula el porcentaje de bloques horarios asignados efectivamente respecto al total disponible.
- Capacidad de Sala (Balance Capacidad/ Estudiantes): Evalúa si la asignación de cursos a salas respeta la capacidad ideal.
- Compactación Horaria (Gaps): Mide el número promedio de huecos (espacios libres) entre clases asignadas a una misma sala o profesor.
- Elegibilidad de Salas (RE-Room Eligibility): Indica cuántas salas cumplen las condiciones mínimas para una actividad específica (por ejemplo, capacidad, tipo de sala, campus). Se expresa como un porcentaje promedio por evento.
- Elegibilidad Temporal (TE-Time Eligibility): Evalúa cuántos períodos horarios son viables para asignar una actividad sin generar conflictos.

RESULTADOS Y DISCUSIÓN

Esta sección muestra y analiza los hallazgos obtenidos a partir de las métricas relacionadas al uso de recursos computacionales y a la calidad de la solución a los tres escenarios de complejidad. La Tabla 1 muestra los escenarios considerados.

Los experimentos se ejecutaron en un computador con procesador Intel Core i7-8750H, 16 GB de RAM y Windows 10 Pro, utilizando JADE 4.5.0 y SPADE 3.4.3. Las métricas de recursos computacionales se recolectaron con el Monitor de Recursos de Windows y registros automáticos, mientras que las métricas de calidad de la solución se obtuvieron a partir de archivos generados por los agentes y analizados mediante scripts.

Uso de recursos computacionales

La Figura 3 muestra el porcentaje de utilización CPU. Los valores bajos (0-5%) indican que ambas plataformas son muy eficientes y no saturan el hardware. JADE tiene un incremento inicial de alrededor 3-4% durante la inicialización, luego se estabiliza en 1%. Esto sugiere una fase de arranque intensiva donde carga recursos o crea agentes, seguida de operación muy liviana. SPADE mantiene un uso constante alrededor del 1% durante toda la ejecución. No presenta incrementos, indicando una arquitectura más uniforme en el consumo de recursos.

La Figura 4 muestra un patrón distintivo en el consumo de memoria entre las plataformas JADE y SPADE que puede explicarse por las características intrínsecas de cada lenguaje y sus respectivos

Tabla 1. Configuración de escenarios.

Escenario	Profesores	Salas
Pequeño	20	15
Mediano	80	40
Completo	153	73

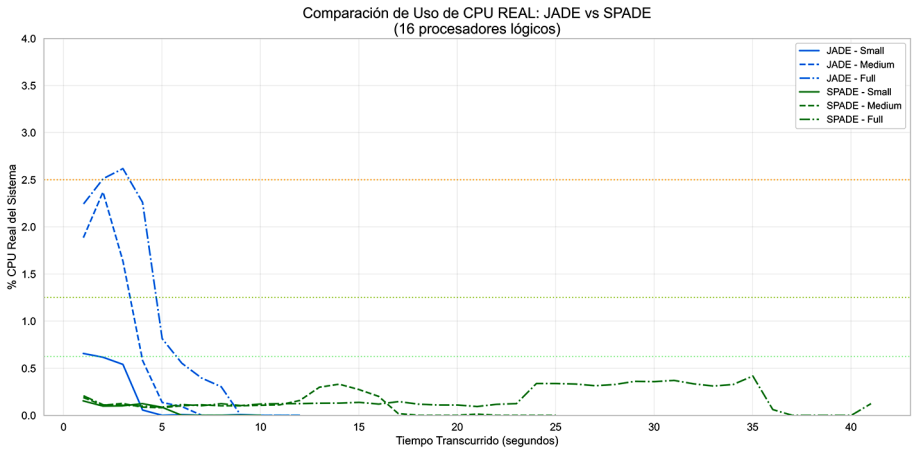

Figura 3. Uso de CPU de ambas plataformas en los tres escenarios.

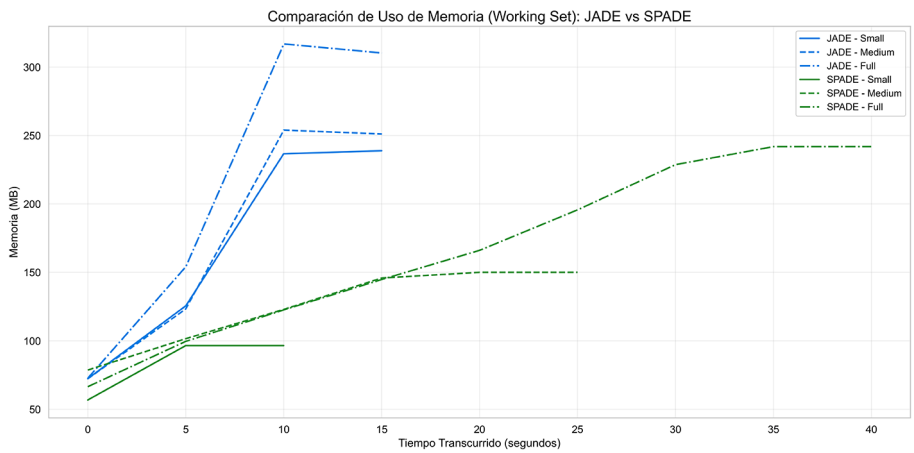

Figura 4. Uso de memoria.

entornos de ejecución. La mayor diferencia se observa en el comportamiento inicial de ambas plataformas. JADE presenta un consumo de memoria base significativamente mayor debido al overhead inherente de la Java Virtual Machine (JVM). Este overhead incluye el heap inicial, el metaspace para definiciones de clases, el code cache, los thread stacks, y las estructuras internas de la JVM. En contraste, SPADE, al ejecutarse directamente sobre el intérprete de Python, tiene un footprint inicial considerablemente menor, comenzando cerca de los 50-70 MB comparado con los aproximadamente 100 MB de JADE.

El tiempo de ida y vuelta (RTT) promedio (Figura 5) favorece a JADE en escenarios de mayor escala, con 1,0 ms frente a 5,2 ms de SPADE. El resultado confirma que la JVM gestiona la serialización de mensajes y la reutilización de hilos con menor latencia que el modelo basado en *asyncio* y XMPP de SPADE cuando el número de agentes supera los 200.

Calidad de la solución

Los porcentajes de ocupación de bloques (Tabla 2) muestran tendencias muy similares entre plataformas: ambos sistemas alcanzan cerca del 48% en el escenario pequeño y rondan el 62% en el escenario grande. La paridad confirma que la lógica de asignación, idéntica en ambas implementaciones, domina este indicador, de modo que la elección de framework no altera la utilización global del espacio.

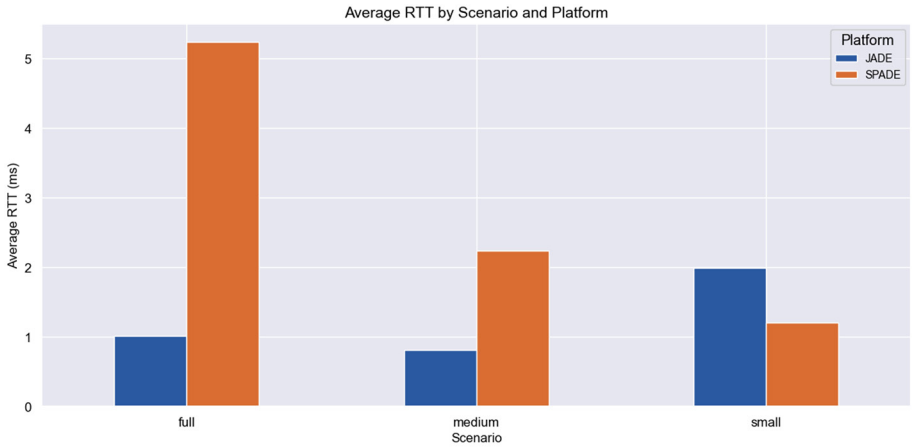

Figura 5. Tiempo de ida y vuelta.

Tabla 2. Ocupación de bloques.

Escenario	Sistema	Bloques Ocupados	Bloques Desocupados	Tasa de Ocupación
Pequeño	JADE	324	351	48,00%
	SPADE	324	351	48,00%
Mediano	JADE	971	829	53,94%
	SPADE	972	828	54,00%
Grande	JADE	2,057	1,228	62,62%
	SPADE	2,048	1,237	62,34%

En cuanto a la capacidad de sala (Tabla 3), ninguna plataforma genera asignaciones que excedan el aforo máximo; más del 65% de los eventos se programan bajo la capacidad nominal y aproximadamente un tercio emplea exactamente la capacidad ideal. SPADE exhibe una ligera ventaja (2 puntos porcentuales) en la correcta utilización de aforo en el escenario mediano, lo que sugiere que su ciclo de propuestas asíncronas encuentra combinaciones marginalmente más ajustadas.

La compactación horaria revela diferencias más marcadas. SPADE reduce el promedio de ventanas por sala a 0,52 en el escenario completo frente a 0,63 de JADE (Tabla 4). Este resultado confirma que el procesamiento basado en *async/await* permite generar propuestas con bloques consecutivos más rápidamente, mejorando la continuidad sin sacrificar otras métricas.

La Tabla 5 muestra las métricas de elegibilidad temporal (TE) y de salas (RE) presentan valores superiores al 87% y del 36%-43% respectivamente, sin diferencias significativas entre plataformas. Esto indica que la disponibilidad de franjas y

Tabla 3. Capacidad de sala.

Escenario	Sistema	Bajo Capacidad	Capacidad Exacta	Sobre Capacidad
Pequeño	JADE	65,12%	34,88%	0,00%
	SPADE	65,12%	34,88%	0,00%
Mediano	JADE	70,75%	29,25%	0,00%
	SPADE	68,42%	31,58%	0,00%
Grande	JADE	66,41%	33,59%	0,00%
	SPADE	67,29%	32,71%	0,00%

Tabla 4. Compactación horaria.

Escenario	Sistema	Ventanas Totales	Promedio Ventanas por Sala
Pequeño	JADE	15	1,00
	SPADE	13	0,87
Mediano	JADE	17	0,42
	SPADE	19	0,47
Grande	JADE	46	0,63
	SPADE	38	0,52

Tabla 5. Elegibilidad temporal y de salas.

Escenario	Sistema	TE Value	TE Percentage
Pequeño	JADE	87,37%	87,37%
	SPADE	87,14%	87,14%
Mediano	JADE	89,53%	89,53%
	SPADE	89,31%	89,31%
Grande	JADE	90,21%	90,21%
	SPADE	89,73%	89,73%

salas adecuadas depende principalmente de las restricciones estructurales del problema más que del motor MAS subyacente.

DISCUSIÓN

El comportamiento diferenciado en el uso de CPU entre JADE y SPADE se explica por las arquitecturas subyacentes de cada plataforma. JADE, al estar implementado en Java y basado en hilos concurrentes, ejecuta múltiples tareas en paralelo aprovechando todos los núcleos disponibles del procesador, lo que genera un consumo elevado de CPU al inicio, asociado a la creación de agentes y al arranque de la JVM. Este consumo decrece rápidamente una vez que los agentes entran en estado de espera o terminan sus tareas. En contraste, SPADE está construido sobre Python y utiliza un modelo de ejecución asíncrono basado en corutinas y el patrón async/await, lo que distribuye la carga de trabajo de forma secuencial y sostenida a lo largo del tiempo. Además, la presencia del Global Interpreter Lock (GIL) en Python impide el paralelismo real, limitando la capacidad de aprovechar múltiples núcleos, lo que resulta en un consumo de CPU más bajo pero constante durante toda la ejecución.

Respecto al uso de memoria, JADE muestra un consumo más alto que SPADE, especialmente a medida que aumenta el número de agentes. Esto se explica por el overhead de la JVM y su gestión interna de objetos e hilos [6]. SPADE, al estar construido sobre Python y utilizar estructuras más ligeras, mantiene un perfil de memoria más eficiente, aunque con un costo en términos de velocidad de ejecución.

En relación con la calidad de la solución, ambas plataformas logran generar asignaciones válidas que cumplen con las restricciones consideradas. Sin embargo, JADE obtiene mejores resultados en indicadores como ocupación de salas, compactación horaria y elegibilidad de bloques y salas, produciendo horarios más eficientes y alineados con las directrices definidas [2]. SPADE, aunque funcional, presenta una mayor dispersión en los bloques asignados y una frecuencia más alta de penalizaciones por traslados entre campus o uso de bloques extremos.

Cabe destacar que estas diferencias se producen a pesar de que el diseño del sistema multiagente es idéntico en ambas plataformas: la lógica de los agentes, las 8 directrices consideradas y el protocolo de comunicación basado en Contract Net Protocol se implementaron de forma equivalente. Las discrepancias en los resultados se explican por diferencias en el comportamiento interno de las plataformas. Por ejemplo, JADE utiliza

múltiples hilos para gestionar comunicaciones y comportamientos de agentes, lo que favorece un procesamiento más rápido y sincronizado de mensajes [20]. SPADE, en cambio, basa su ejecución en un modelo asincrónico que puede introducir retardo en la gestión de propuestas y afectar el orden de recepción de mensajes, especialmente en contextos con alta simultaneidad de negociaciones [15].

Además, pequeños desfases en la activación de ciclos, el manejo de colas de mensajes o la resolución de empates pueden llevar a divergencias en el proceso de asignación, aun cuando las condiciones iniciales sean iguales. Esto muestra que, si bien la equivalencia conceptual del sistema garantiza coherencia funcional, la arquitectura de ejecución de cada plataforma puede influir en la dinámica de interacción y, por ende, en la calidad final de la solución.

En síntesis, JADE se comporta mejor en términos de eficiencia computacional y calidad de solución bajo el diseño de agentes utilizado, mientras que SPADE, aunque más flexible y modular, requiere mayores recursos y presenta una menor consistencia en los indicadores clave. Esta diferencia resulta relevante al momento de elegir una plataforma para escenarios reales donde el rendimiento y la precisión de la planificación son factores críticos.

CONCLUSIONES

Este estudio demostró la viabilidad de emplear sistemas multiagente para la generación automática de horarios académicos y comparó, bajo un mismo caso de estudio, las plataformas JADE y SPADE. La arquitectura basada en el protocolo Contract-Net permitió modelar la interacción entre agentes profesor y sala de forma transparente, reproduciendo dieciséis restricciones institucionales con resultados estables en los tres escenarios de complejidad.

Los experimentos revelan que ambas plataformas entregan horarios de calidad semejante; sin embargo, presentan perfiles de rendimiento claramente diferenciados. JADE exhibe menor latencia y un consumo de recursos más concentrado, cualidades que lo hacen apto para entornos de alta carga y requisitos de respuesta strictos. SPADE, por su parte, muestra un uso de memoria inicial inferior y genera horarios más compactos, lo que lo convierte en una opción competitiva para implementaciones de menor escala o donde la continuidad horaria sea prioritaria.

Como líneas de trabajo futuro se propone incorporar técnicas de aprendizaje por refuerzo para que los agentes ajusten dinámicamente sus estrategias de negociación, evaluar otras plataformas como MadKit, Jadex o Jason bajo las mismas métricas y analizar la robustez del sistema ante fallos de red o modificaciones abruptas en las restricciones. También se sugiere explorar la integración con herramientas de visualización en tiempo real que faciliten la validación de los horarios por parte de los usuarios finales.

REFERENCIAS

[1] P. Brucker, *Scheduling algorithms*, 5th ed. New York, USA: Springer, 2007, doi: 10.1007/978-3-540-69516-5.

[2] S. Ceschia, L. Di Gaspero, and A. Schaerf, "Educational timetabling: Problems, benchmarks, and state-of-the-art results," *European Journal of Operational Research*, vol. 308, no. 1, pp. 1-18, 2023, doi: 10.1016/j.ejor.2022.07.011.

[3] E.K. Burke and S. Petrovic, "Recent research directions in automated timetabling," *European Journal of Operational Research*, vol. 140, no. 2, pp. 266-280, 2002, doi: 10.1016/S0377-2217(02)00069-3.

[4] M.J. Wooldridge, *An Introduction to Multiagent SystemsAn Introduction to Multiagent*, 2nd ed. London, UK: Wiley, 2009.

[5] R. Leszczyna, "Evaluation of Agent Platforms (ver. 2.0)," Joint Research Centre(JRC), Luxembourg, JRC47224, 2008. Accessed: Nov. 5, 2025. [Online]. Available: https://publications.jrc.ec.europa.eu/repository/handle/JRC47224

[6] Z. Wrona *et al.*, "Overview of Software Agent Platforms Available in 2023," *Information*, vol. 14, no. 6, Art. no. 348, 2020, doi: 10.3390/info14060348.

[7] D. Landa-Silva and J.H. Obit, "Evolutionary non-linear great deluge for university course timetabling," in *Hybrid Artificial Intelligence Systems. HAIS 2009*, E. Orchado, X. Wu, E. Oja, Á. Herrero,

and B. Baruque, Eds., vol. 5572, Berlin, Germany: Springer,2009, pp. 269-276, doi: 10.1007/978-3-642-02319-4_32.

[8] M. Chiarandini, L. Di Gaspero, S. Gualandi, and A. Schaerf, "The balanced academic curriculum problem revisited," *Journal of Heuristics*, vol. 18, no. 1, pp. 119-148, 2012, doi: 10.1007/s10732-011-9158-2.

[9] A. Lemos, F.S. Melo, P.T. Monteiro, and I. Lynce, "Room usage optimization in timetabling: A case study at Universidade de Lisboa," *Operations Research Perspectives*, vol. 6, Art. no. 100092, 2019, doi: 10.1016/j.orp.2018.100092.

[10] A.D. Guia and M.A. Ballera, "Multi-agent class timetabling for higher educational institutions using prometheus platform," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 22, no. 3, pp. 1679-1687, Jun. 2021, doi: 10.11591/ijeecs.v22.i3. pp. 1679-1687.

[11] Z. Houhamdi, B. Athamena, R. Abuzaineddin, and M. Muhairat, "A multi-agent system for course timetable generation," *Technology Education Management Informatics*, vol. 8, no. 1, pp. 211-221, 2019, doi: 10.18421/TEM81-30.

[12] M. Oprea, "MAS_UP-UCT: A multi-agent system for university course timetable scheduling," *International Journal of Computers, Communications & Control*, vol. 2, no. 1, pp. 94-102, 2007.

[13] C. Covantes and R. Rodríguez, "Design of multi-agent system for solution of the school timetabling problem," in *2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI)*, Cuernavaca, Mexico, Oct. 25-31 2015, pp. 173-181, doi: 10.1109/ MICAI.2015.33.

[14] A. Siam, S.M. El Habib, and S. Safir, "A multi-agent System for Generation of University Time Schedules," in *8th International Conference on Advanced Computer Science and Information Technology (ICAIT 2019)*, N. Meghanathan and D.C. Wyld, Eds., Zurich, Switzerland, Mar.30-31, 2019, doi: 10.5121/csit.2019.90403.

[15] J. Palanca, A. Terrasa, V. Julian, and C. Carrascosa, "Spade 3: Supporting the new generation of multi-agent systems," *IEEE Access*, vol. 8, pp. 182537-182549, 2020, doi: 10.1109/ACCESS.2020.3027357.

[16] D. Mrozek, B. Małysiak-Mrozek, and I. Waligóra, "UMAP - A Universal Multi-Agent Platform for .NET Developers," in *Beyond Databases, Architectures, and Structures*, S. Kozielski, D. Mrozek, P. Kasprowski, B. Małysiak-Mrozek, and D. Kostrzewa, Eds., Cham, Switzerland: Springer, 2014, pp. 300-311, doi: 10.1007/978-3-319-06932-6_29.

[17] G. Radhakrishnan, V. Chithambaram, and K.L. Shunmuganathan, "Comparative Study of Jade and Spade Multi Agent System," *International Journal of Advanced Research,* vol. 6, no. 11, pp. 1035-1042, 2018, doi: 10.21474/IJAR01/8090.

[18] Z. Wrona *et al.*, "Comparison of Multi-Agent platform usability for Industrial-Grade applications," *Applied Sciences*, vol. 14, no. 22, Art. no. 10124, 2024, doi: 10.3390/app142210124.

[19] R.G. Smith, "The Contract Net Protocol: High-level communication and control in a distributed problem solver," *IEEE Transactions on Computers*, vol. C-29, no. 12, pp. 1104-1113, 1980, doi: 10.1109/TC.1980.1675516.

[20] F. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. Chichester, UK: John Wiley & Sons, 2007, doi: 10.1002/9780470058411.

[21] S. Dähling, L. Razik, and A. Monti, "Enabling scalable and fault-tolerant multi-agent systems by utilizing cloud-native computing," *Autonomous Agents and Multi-Agent Systems*, vol. 35, no. 10, 2021, doi: 10.1007/s10458-020-09489-0.