

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“DISEÑO E INTEGRACIÓN DE UN MOTOR DE REGLAS
FLEXIBLES PARA UN ALGORITMO DE PLANIFICACIÓN:
UN ESTUDIO EN LA EMPRESA UPLANNER.”

THEO JOFRÉ

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: María Cristina Riff
Profesor Correferente: Camila Díaz

Febrero - 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Diseño e integración de un motor de reglas flexibles para un algoritmo de planificación:
Un estudio en la empresa uPlanner.

Nombre del candidato(a): Theo Leviatham Jofré Machuca

Carrera / Grado: Ingeniería Civil Informática

Campus: Casa Central Valparaíso **Departamento:** Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, María Cristina Riff _____, en mi calidad de profesor(a)
guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 17 Junio 2026 **Firma:** _____

Estudiante o Candidato(a):

Fecha: 10-3-2026 **Firma:** _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

Para mi marido, por darme todo el amor y la paciencia que necesitaba.

Para mi abuelo, por ser el primer ingeniero en mi vida.

Para mi hermano, por todo el apoyo que me dio a lo largo de la carrera.

AGRADECIMIENTOS

Quiero agradecer a todas las personas que estuvieron para mí a lo largo de esta más de media década.

Gracias Gabe por ser mi amorcito, mi compañero de vida, ser mi todo y muchas gracias por absolutamente todo lo que me das en mi día a día.

Gracias Profe Riff por ayudarme a encontrar la práctica que hizo posible esta memoria. Gracias Cami por ser mi guía en uPlanner y estar conmigo a lo largo de este proceso. Gracias Bárbara por ayudarme con el inicio de mi viaje a una mejor salud mental.

Gracias Dante y Rorro por ser ustedes y ser unos reales conmigo siempre. Gracias Roberto por apreciar la música juntos. Gracias Paskual por apoyarme en mi tiempo de USACH. Gracias Mathi por ser un amor tan grande en mi vida. Gracias Caro por entenderme y quererme a lo largo de los años. Gracias Rueda e Isidoro por ser los mejores collicanos del mundo. Gracias Franco por ser el número 1. Gracias Pato for being there to fight the sadness. Gracias Isi por todos los años juntos. Gracias Hermanos por nunca verme en menos por ser raro. Gracias Legio por el apoyo que me dieron en el momento que lo necesité. Gracias Sofi por el apañe. Gracias Arielox, Benja y Charlie por mantener la amistad pese al peso de los años. Gracias Yanko por ser un real y porque gracias a ti me di cuenta de que soy un horologium. Gracias Rubén y Vitoko por ser ustedes.

Gracias Tata y Wela por todo el apoyo a lo largo de los años. Gracias padre, porque por cada paso que di siempre pensé en ti. Gracias madre.

Gracias a los ciudadanos de Rebornia, Balatria y Registro Civil de Collico por estar ahí para mí. Gracias juegos de pelea por ser cool AF. Gracias al club de los cojos por estar ahí conmigo pasando los ramos juntos. Gracias Denyss por tu apañe. Gracias Ardix por los tiempos que hemos estado juntos a lo largo de los años.

Gracias a todos los mencionados y a los que no, por ser parte de mi vida y apoyarme en este proceso.

RESUMEN

Resumen— Se diseñó e implementó un prototipo funcional de Motor de Reglas de Negocio (BRE) para el sistema de planificación de exámenes de uPlanner, separando la lógica de restricciones institucionales del algoritmo heurístico mediante un lenguaje específico de dominio (DSL) declarativo basado en JSON. El motor, desarrollado en Python, incluye 18 operadores reutilizables, evaluación individual y grupal, condiciones de aplicabilidad selectiva, y un índice de grupos invertido que reduce la complejidad de evaluación en lote de $O(n^2)$ a $O(n)$. Validado sobre 4.557 propuestas reales con 24 reglas, alcanzó un 67 % de tasa de cumplimiento global con 10 reglas en 100 %, e identificó brechas sistemáticas en distribución de evaluaciones. La solución demuestra la viabilidad de declarar y auditar restricciones académicas sin intervención del equipo de desarrollo.

Palabras Clave— Motor de reglas de negocio; lenguaje específico de dominio; planificación de exámenes; evaluación de restricciones académicas; arquitectura desacoplada.

ABSTRACT

Abstract— A functional Business Rules Engine (BRE) prototype was designed and implemented for uPlanner's exam scheduling system, decoupling institutional constraint logic from the heuristic assignment algorithm through a declarative JSON-based domain-specific language (DSL). The engine, built in Python, features 18 reusable operators, individual and group-scope evaluation, selective applicability conditions, and an inverted group index reducing batch evaluation complexity from $O(n^2)$ to $O(n)$. Validated on 4,557 real proposals against 24 rules, it achieved a 67 % global compliance rate with 10 rules at 100 % compliance, and identified systematic gaps in exam distribution constraints. The solution demonstrates the feasibility of declaratively defining and auditing academic constraints without developer intervention.

Keywords— Business rules engine; domain-specific language; exam scheduling; academic constraint evaluation; decoupled architecture.

GLOSARIO

API (*Application Programming Interface*): Interfaz de programación de aplicaciones. Conjunto de definiciones y protocolos que permite la comunicación entre sistemas de software.

BRE (*Business Rules Engine*): Motor de Reglas de Negocio. Sistema de software que ejecuta reglas de negocio separadas del código de la aplicación principal.

CSV (*Comma-Separated Values*): Formato de archivo de texto plano que almacena datos tabulares separados por comas.

DMN (*Decision Model and Notation*): Estándar de notación para modelar decisiones de negocio, publicado por la *Object Management Group* (OMG).

DSL (*Domain-Specific Language*): Lenguaje específico de dominio. Lenguaje de programación o especificación diseñado para un dominio particular de aplicación.

FastAPI: Framework de Python de alto rendimiento para la construcción de APIs REST, basado en anotaciones de tipos estándar.

HTTP (*HyperText Transfer Protocol*): Protocolo de transferencia de hipertexto utilizado para la comunicación en la web.

JSON (*JavaScript Object Notation*): Formato ligero de intercambio de datos basado en texto, legible por humanos y fácilmente procesable por máquinas.

N/A (*No Aplicable*): Estado de validez que indica que una regla no corresponde a la propuesta evaluada debido a sus condiciones de aplicabilidad.

NP-completo: Clase de problemas de decisión para los cuales no se conoce un algoritmo de solución en tiempo polinomial. Los problemas de planificación académica se clasifican en esta categoría.

ORM (*Object-Relational Mapping*): Técnica de programación que convierte datos entre sistemas de tipos incompatibles en lenguajes de programación orientados a objetos y bases de datos relacionales.

Pydantic: Biblioteca de Python para validación de datos y gestión de configuraciones mediante anotaciones de tipos, utilizada en este trabajo para la definición del modelo de datos.

REST (*Representational State Transfer*): Estilo de arquitectura de software para sistemas hipermidia distribuidos, ampliamente utilizado en el diseño de APIs web.

Streamlit: Framework de Python de código abierto para la creación rápida de aplicaciones web interactivas orientadas a datos.

uPlanner: Empresa de tecnología educativa que desarrolla soluciones de planificación académica para instituciones de educación superior en Latinoamérica.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	VIII
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	3
1.1 Contexto del problema	3
1.2 Objetivos y alcance	4
1.2.1 Objetivo general	4
1.2.2 Objetivos específicos	4
1.2.3 Alcance	5
CAPÍTULO 2: Revisión de la literatura	7
3 Marco conceptual	7
3.1 Business Rule Engines (BRE) y Sistemas Basados en Reglas(SBR)	7
3.2 Arquitectura y Componentes Fundamentales	7
3.3 Clasificación y Tipos de Motores de Reglas	9
4 Estado del arte	10
4.1 Plataformas de BRE Principales	10
4.2 Aplicación de BRE en Sistemas de Scheduling	10
4.3 Casos de Estudio Específicos para Scheduling Académico	12
CAPÍTULO 5: PROPUESTA DE SOLUCIÓN	13
5.1 Visión General de la Solución	13
5.2 Evolución del Modelo de Datos	13
5.3 Carga de Datos desde Archivos CSV	16
5.4 Lenguaje de Definición de Reglas Declarativo	17
5.4.1 Estructura de una Regla	17
5.4.2 Resolución de Hechos mediante Notación de Puntos	19
5.4.3 Alcance de Evaluación: Sección y Grupo	19
5.4.4 Condiciones de Aplicabilidad	20
5.5 Arquitectura e Implementación del Motor de Reglas	20
5.6 Diseño del Conjunto Mínimo de Operadores	21
5.7 Optimizaciones de Rendimiento	22

5.8	Interfaces de la Solución	24
5.8.1	API REST con FastAPI	24
5.8.2	Interfaz Web con Streamlit	25
5.9	Catálogo de Reglas Implementadas	29
CAPÍTULO 6: VALIDACIÓN DE LA SOLUCIÓN		32
6.1	Metodología de Validación	32
6.2	Descripción del Conjunto de Datos	32
6.3	Resultados de la Evaluación en Lote	33
6.4	Análisis de Resultados	34
6.4.1	Reglas con Cumplimiento Total	34
6.4.2	Restricciones con Mayor Tasa de Incumplimiento	35
6.4.3	Reglas con Condiciones de Aplicabilidad y Bajo Cumplimiento	36
6.4.4	Verificación del Mecanismo de Aplicabilidad	37
6.5	Verificación de los Criterios de Validación	37
CAPÍTULO 7: CONCLUSIONES		39
7.1	Cumplimiento de Objetivos	39
7.2	Contribuciones Principales	40
7.3	Resultados de la Validación	40
7.4	Limitaciones y Trabajo Futuro	41
7.5	Reflexión Final	41
ANEXOS		42
REFERENCIAS BIBLIOGRÁFICAS		45

ÍNDICE DE FIGURAS

1	Árbol de problemas del sistema de planificación académica.	3
2	Arquitectura básica y ciclo de ejecución de un Business Rule Engine.	8
3	Arquitectura general de la solución propuesta. Elaboración propia.	14
4	Ciclo de evaluación de una regla individual en el motor. Elaboración propia. .	21
5	Página <i>Crear Regla</i> : formulario de dos pasos para definir condiciones y guardar una nueva regla en <code>rules.json</code> . Elaboración propia.	25
6	Página <i>Probar Reglas</i> (a): selector de propuesta y panel desplegable con los atributos del objeto <code>schedules</code> seleccionado. Elaboración propia.	26
7	Página <i>Probar Reglas</i> (b): tabla de resultados de evaluación por regla, con indicador de estado y mensaje de error cuando corresponde. Elaboración propia.	26
8	Página <i>Probar Reglas</i> (c): continuación de la tabla de resultados y respuesta JSON completa del motor de reglas. Elaboración propia.	27
9	Página <i>Listar Reglas</i> : catálogo de las 24 reglas configuradas, con nombre e identificador, desplegables para auditar condiciones individuales. Elaboración propia.	27
10	Página <i>Probar Filtrado de Opciones</i> : entrada JSON de opciones candidatas (salas, docentes o calendarios) y botón de filtrado que invoca <code>filter_options</code> . Elaboración propia.	28
11	Página <i>Resumen de Reglas</i> (a): métricas globales (4.557 horarios evaluados, promedio de 18,1/24 reglas válidas) y gráfico de barras apilado con distribución de estados por regla. Elaboración propia.	28
12	Página <i>Resumen de Reglas</i> (b): tabla paginada de horarios individuales con estado por regla (<i>válido</i> , <i>N/A</i> e <i>inválido</i>), filtrable y exportable a CSV. Elaboración propia.	29
13	Página <i>Resumen de Reglas</i> (c): tabla de totales por regla con conteo de válidos, <i>N/A</i> e inválidos sobre la totalidad de las 4.557 propuestas evaluadas. Elaboración propia.	29
14	Modelo de datos original del dominio de planificación académica. Elaboración propia.	43
15	Modelo de datos actual implementado con Pydantic. Elaboración propia. . .	44

ÍNDICE DE TABLAS

1	Comparativa de Principales Plataformas de Business Rule Engines	11
2	Secuencia de carga de datos desde archivos CSV.	16
3	Conjunto de operadores implementados en el motor de reglas.	23
4	Endpoints de la API REST del motor de reglas.	24
5	Catálogo de reglas de negocio implementadas en el motor de reglas. Elaboración personal	29

6	Resultados de la evaluación en lote: 4.557 propuestas y 24 reglas (109.368 evaluaciones).	33
---	---	----

INTRODUCCIÓN

La planificación académica en instituciones de educación superior constituye uno de los problemas combinatoriales más complejos del ámbito de la gestión universitaria. En su formulación general, el problema consiste en asignar un conjunto de secciones académicas a combinaciones de franjas horarias, salas físicas o virtuales y docentes, satisfaciendo simultáneamente un amplio conjunto de restricciones institucionales: disponibilidades de docentes y aulas, capacidades físicas, contratos laborales, normativas de distribución de evaluaciones, preferencias de programación, entre otras. Este tipo de problema pertenece a la clase de complejidad NP-completo [Russell y Norvig, 2020], lo que implica que no existe un algoritmo capaz de resolverlo de forma exacta en tiempo polinomial para instancias de tamaño real. La resolución práctica, por tanto, depende de algoritmos heurísticos que aproximan soluciones de alta calidad dentro de tiempos acotados.

La empresa uPlanner provee soluciones de planificación académica para universidades de toda Latinoamérica. Su plataforma incorpora un algoritmo heurístico de asignación que opera sobre los datos de cada institución cliente, generando propuestas que deben respetar las políticas académicas vigentes. Dado que cada institución define sus propias reglas de acuerdo a su reglamento interno, el algoritmo debe ser adaptado continuamente a medida que dichas políticas evolucionan. Esta necesidad de adaptación representa un desafío técnico y operativo permanente para el equipo de desarrollo de uPlanner.

El problema concreto que motiva la presente memoria es que, en el sistema actual de uPlanner, las restricciones institucionales se encuentran *embebidas directamente en el código* del algoritmo de asignación. Las políticas académicas —como el máximo de evaluaciones por día por grupo de estudiantes, la modalidad de asignación de supervisores o las reglas de programación en bloques específicos— no están expresadas de forma separada ni declarativa, sino entrelazadas con la lógica de búsqueda heurística. Como consecuencia, cada vez que una institución modifica o introduce una política, un desarrollador debe intervenir el código fuente, recompilar el sistema, ejecutar pruebas de regresión y redespargar la solución. Este ciclo de cambio es lento, costoso y propenso a errores, limitando significativamente la capacidad de uPlanner de responder con agilidad a la dinámica institucional de sus clientes.

La solución propuesta en esta memoria consiste en diseñar e implementar un *Motor de Reglas de Negocio* (BRE, del inglés *Business Rules Engine*) como capa de validación desacoplada del algoritmo heurístico. El principio arquitectónico central es la separación entre dos tipos de lógica: la *lógica de negocio volátil*, constituida por las restricciones académicas que varían por institución y en el tiempo, y la *lógica computacional estable*, que implementa el mecanismo de evaluación y permanece inalterada. Las restricciones se expresan de forma declarativa en un archivo de configuración JSON mediante un Lenguaje Específico de Dominio (DSL) diseñado para este propósito, de modo que un administrador académico o un analista con conocimientos básicos de programación pueda agregar, modificar o desactivar reglas sin necesidad de tocar el código del motor.

Si bien el motor fue diseñado con flexibilidad suficiente para aplicarse al dominio general de planificación académica, la implementación prototípica desarrollada en esta memoria se centra específicamente en la **asignación de evaluaciones académicas (exámenes)**, que constituye un caso de uso simplificado pero representativo del problema más amplio de planificación horario-sala-docente. Este alcance permite demostrar la viabilidad del enfoque con datos reales de producción y sentar las bases para una futura generalización al dominio completo.

El desarrollo de la solución siguió una metodología iterativa organizada en cuatro etapas: (1) modelado del dominio mediante diez clases Pydantic que representan fielmente las entidades involucradas en una asignación de examen; (2) diseño del DSL y del motor de evaluación, con soporte para 18 operadores reutilizables en cinco categorías, evaluación de alcance individual y grupal, y optimizaciones de caché que reducen la complejidad de evaluación en lote; (3) exposición del motor como microservicio REST mediante FastAPI y como aplicación web interactiva mediante Streamlit, que habilitan su uso tanto en integración con el algoritmo como por usuarios no técnicos; y (4) validación empírica sobre 4.557 propuestas de asignación reales generadas por el sistema de uPlanner, evaluadas contra un catálogo de 24 reglas que produce 109.368 evaluaciones individuales.

El presente documento se organiza de la siguiente manera. El Capítulo 2 presenta la definición del problema, los objetivos y el alcance del trabajo. El Capítulo 3 desarrolla el marco conceptual, revisando los fundamentos teóricos de los motores de reglas de negocio, los lenguajes específicos de dominio y los enfoques computacionales para la planificación académica. El Capítulo 4 describe en detalle la propuesta de solución: el modelo de datos, el diseño del DSL, la arquitectura del motor de evaluación, las interfaces de usuario y programáticas desarrolladas, y el catálogo de 24 reglas implementadas. El Capítulo 5 presenta la validación de la solución y el análisis cuantitativo de los resultados obtenidos. Finalmente, el Capítulo 6 expone las conclusiones, las contribuciones técnicas de la memoria, las limitaciones identificadas y las líneas de trabajo futuro.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

1.1. Contexto del problema

La planificación académica en instituciones de educación superior representa un desafío logístico complejo debido a la alta demanda de recursos limitados como salas de clases, horarios disponibles y personal docente. La empresa uPlanner provee soluciones de planificación para este problema al adaptar su algoritmo heurístico a las reglas específicas y dinámicas de cada institución educativa. La rigidez del sistema actual, donde las reglas están embebidas en el código, impide una modificación dinámica y ágil de las restricciones académicas, generando ineficiencias operativas y afectando la satisfacción de los usuarios finales (generalmente administradores académicos y personal docente).

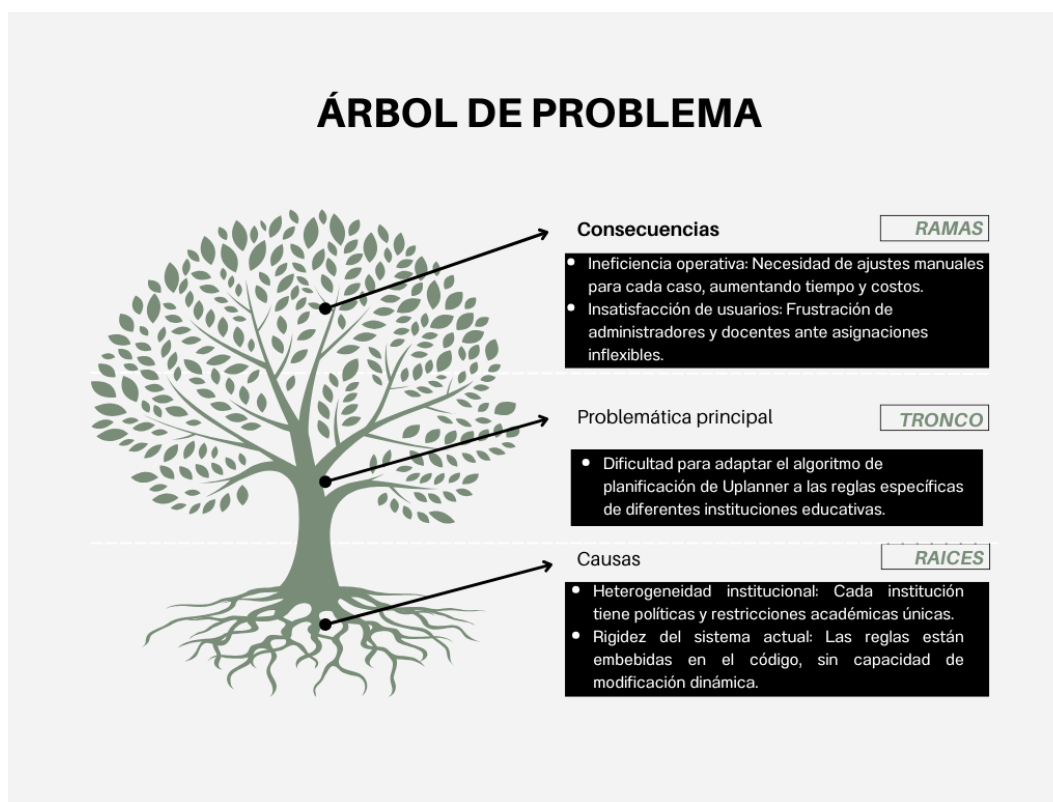


Figura 1: Árbol de problemas del sistema de planificación académica.

La complejidad computacional de estos problemas, clasificados como NP-completos [Russell y Norvig, 2020], exige enfoques que equilibren la eficiencia algorítmica con la flexibilidad para incorporar restricciones específicas del dominio académico. Revisiones sistemáticas del estado del arte en planificación académica coinciden en que los sistemas exitosos deben

permitir la adaptación dinámica de políticas y reglas institucionales [Burke y Petrovic, 2002, Qu *et al.*, 2009].

La arquitectura de Motores de Reglas de Negocio (BRE) emerge como una solución viable para este desafío, permitiendo separar la lógica de las reglas del código del algoritmo de planificación. Este principio de desacoplamiento, fundamental en el diseño de lenguajes específicos de dominio [Fowler, 2010], facilitaría que expertos académicos modifiquen políticas sin depender de ciclos de desarrollo de software. Investigaciones recientes han documentado aplicaciones exitosas de BRE en entornos educativos, demostrando mejoras significativas en adaptabilidad y transparencia de los procesos de planificación [Vanthienen y Caron, 2013, Alvarez *et al.*, 2018].

En resumen, la planificación académica puede beneficiarse de la integración de un motor de reglas flexible que permita definir, priorizar y gestionar restricciones de manera declarativa, mejorando la adaptabilidad del sistema, la eficiencia operativa y la transparencia del proceso de asignación.

1.2. Objetivos y alcance

El objetivo de esta memoria es diseñar e integrar un motor de reglas flexible y extensible que optimice el algoritmo de planificación utilizado por la empresa uPlanner, mejorando la capacidad del sistema para adaptarse a las reglas específicas de diversas instituciones de educación superior.

1.2.1. Objetivo general

Diseñar e implementar un motor de reglas flexible y extensible que permita la definición, gestión y evaluación de restricciones académicas de forma declarativa. Así, desacoplando la lógica de negocio del algoritmo de planificación para facilitar el mantenimiento, la escalabilidad y la adaptación del sistema de asignación de evaluaciones académicas (exámenes) a las variables horario-sala-docente en instituciones de educación superior.

1.2.2. Objetivos específicos

1. Diseñar un lenguaje de definición de reglas declarativo que permita migrar y expresar las reglas preexistentes del sistema heurístico actual, así como definir nuevas restricciones académicas (disponibilidad docente, capacidad de aulas, conflictos horarios, etc.) de manera intuitiva y estructurada, sin requerir modificaciones en el código fuente del sistema. Para esto, se seguirán principios de diseño de lenguajes específicos de

dominio [Fowler, 2010] y se considerarán estándares como DMN [Object Management Group, 2018] para asegurar claridad y mantenibilidad.

2. Crear una arquitectura extensible y configurable que permita la adición, modificación y eliminación de reglas en tiempo de ejecución, manteniendo la compatibilidad con reglas existentes y facilitando la incorporación de nuevos tipos de restricciones académicas. La arquitectura se basará en patrones establecidos para sistemas basados en reglas [Von Halle, 2001].
3. Establecer un mecanismo de priorización y categorización de reglas que permita clasificar las restricciones por importancia (obligatorias, preferibles, opcionales) y resolver conflictos entre reglas de manera consistente y predecible. Este mecanismo incorporará estrategias de resolución de conflictos bien estudiadas en sistemas basados en reglas [Giarratano y Riley, 2005, Ross, 2013].
4. Implementar un sistema de evaluación de reglas modular que pueda procesar y validar múltiples tipos de restricciones de forma eficiente, proporcionando retroalimentación detallada sobre violaciones y su severidad. La evaluación debe ser suficientemente eficiente para integrarse dentro de un algoritmo de búsqueda heurística, considerando las lecciones de integración entre BRE y solvers de planificación documentadas en la literatura [Muller y Rudova, 2005, Zhao *et al.*, 2019].
5. Integrar el motor de reglas con interfaces de consulta y validación que faciliten su uso tanto en algoritmos de planificación automatizada como en sistemas de validación manual. Se buscará proporcionar retroalimentación explicativa, similar a la propuesta en trabajos sobre sistemas de planificación adaptativos [Li *et al.*, 2020], para apoyar la toma de decisiones.

1.2.3. Alcance

El desarrollo de esta memoria se centrará en el diseño e integración del motor de reglas con el algoritmo heurístico existente de uPlanner. Se considerarán las restricciones académicas típicas documentadas en la literatura, como disponibilidad de recursos, preferencias docentes y normativas institucionales [Alvarez *et al.*, 2018, Garcia *et al.*, 2021]. La implementación se realizará mediante prototipos que validen la funcionalidad y eficacia del motor, sin incluir el desarrollo de un sistema de planificación completo desde cero. El estudio se limitará al contexto de instituciones de educación superior y no abordará otros dominios de planificación.

Cabe destacar que, si bien el motor de reglas fue diseñado con la flexibilidad suficiente para aplicarse al dominio general de planificación académica, la implementación prototípica realizada en esta memoria se centra específicamente en la **asignación de evaluaciones académicas (exámenes)**, que representa un caso de uso simplificado pero representativo del

problema general de planificación horario-sala-docente. Este recorte de alcance permite validar el motor sobre datos reales sin abordar la complejidad adicional de la planificación de clases regulares, la cual involucra restricciones de recurrencia semanal y mayor diversidad de tipos de sección.

CAPÍTULO 2

Revisión de la literatura

Marco conceptual

Esta sección presenta los fundamentos teóricos de los Sistemas Basados en Reglas y los Motores de Reglas de Negocio (Business Rule Engines, BRE), definiendo su arquitectura, componentes y principios de funcionamiento, que constituyen la base de la solución propuesta.

3.1. Business Rule Engines (BRE) y Sistemas Basados en Reglas(SBR)

Un Motor de Reglas de Negocio (BRE) es un sistema de software diseñado para gestionar y ejecutar lógica de decisión empresarial de forma independiente al código central de una aplicación. Su propósito principal es permitir a las organizaciones definir, implementar, auditar y mantener las *reglas de negocio* —es decir, las políticas, condiciones, restricciones y procedimientos que guían las operaciones y decisiones empresariales— en un repositorio centralizado y accesible. Este enfoque estratégico separa la lógica de negocio, que es dinámica y propensa a cambios, del código de programación estático. Así ofrece una agilidad operativa significativa [Von Halle, 2001]. En la práctica, esto significa que expertos del dominio o administradores pueden modificar, añadir o desactivar reglas sin necesidad de realizar costosos y riesgosos cambios en el código base del sistema, reduciendo el tiempo de implementación y minimizando errores.

Tecnológicamente, los BRE evolucionan a partir de los Sistemas Basados en Reglas (SBR), un paradigma fundamental en la Inteligencia Artificial que utiliza reglas condicionales del tipo *SI (condición) ENTONCES (acción)* como unidad básica de conocimiento. A diferencia de un programa con un flujo secuencial predefinido, la ejecución en un SBR es dirigida por los datos. Por lo tanto, el motor evalúa continuamente el estado del sistema (representado por un conjunto de *hechos*) contra el conjunto de reglas, ejecutando aquellas cuyas condiciones se cumplan. Esta separación entre conocimiento (reglas) y control (motor de inferencia) es la piedra angular de su flexibilidad [Giarratano y Riley, 2005].

3.2. Arquitectura y Componentes Fundamentales

La arquitectura de un BRE típico se organiza en tres componentes principales que interactúan durante un ciclo de ejecución, tal como se ilustra en la Figura 2.

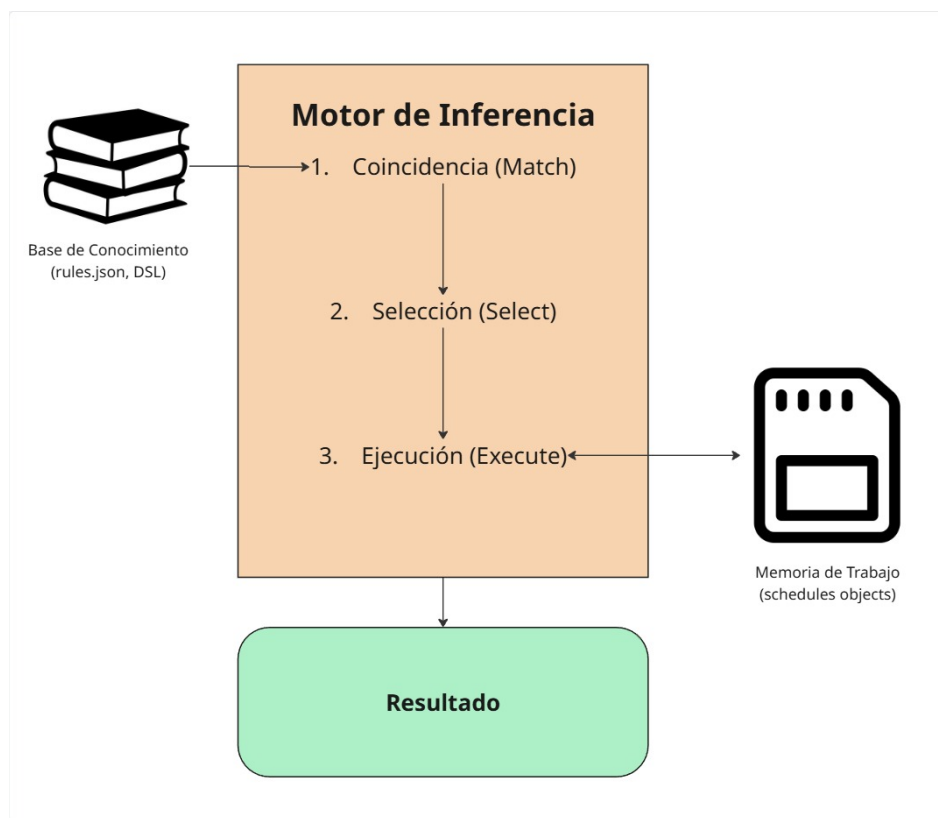


Figura 2: Arquitectura básica y ciclo de ejecución de un Business Rule Engine.

- **Base de Conocimiento (Knowledge Base):** Es el almacén donde se guardan las reglas de negocio. Estas pueden estar escritas en un lenguaje natural estructurado, un lenguaje específico de dominio (DSL) o en un formato técnico propio del motor [Fowler, 2010]. La base de conocimiento es independiente de la aplicación y, en sistemas avanzados, puede ser gestionada a través de interfaces amigables para usuarios no técnicos.
- **Memoria de Trabajo (Working Memory):** Es un área de memoria volátil que contiene los *hechos* o datos de entrada sobre los que se evaluarán las reglas. Estos hechos son representaciones de los objetos del dominio (Por ejemplo: Profesor, Sala, Curso) con sus atributos y estados actuales.
- **Motor de Inferencia (Inference Engine):** Es el núcleo inteligente del sistema. Su función es controlar el ciclo de ejecución aplicando las reglas a los hechos presentes en la memoria de trabajo. Este ciclo se divide en tres fases iterativas:
 1. **Coincidencia (Match):** Compara todas las condiciones (lado izquierdo) de las reglas en la base de conocimiento con los hechos actuales en la memoria de trabajo. El resultado es un conjunto de reglas instanciadas o *activadas*, listas para ejecutarse.

2. **Selección (Select):** Entre todas las reglas activadas, elige cuál será ejecutada a continuación. Dado que múltiples reglas pueden activarse simultáneamente, el motor utiliza una *estrategia de resolución de conflictos* (Por ejemplo: prioridad, novedad, especificidad) para tomar esta decisión de manera determinista [Ross, 2013].
3. **Ejecución (Execute):** Ejecuta las acciones (lado derecho) de la regla seleccionada. Estas acciones pueden modificar la memoria de trabajo (añadiendo, modificando o eliminando hechos), lo que a su vez puede provocar nuevas activaciones de reglas, reiniciando el ciclo.

3.3. Clasificación y Tipos de Motores de Reglas

Los BRE pueden clasificarse según su estrategia de ejecución o inferencia, lo que determina su idoneidad para diferentes tipos de problemas:

- **Encadenamiento hacia adelante (Forward Chaining / Data-Driven):** Parte de los hechos disponibles (los datos) y aplica las reglas para derivar nuevas conclusiones o hechos, hasta que no se puedan inferir más. Es el tipo más común y se utiliza en sistemas donde se necesita explorar todas las consecuencias de un conjunto de datos de entrada. Los motores de **producción** (clásicos) y los **reactivos** o de eventos (ECA o Event-Condition-Action) pertenecen a esta categoría. Herramientas como Drools [Red Hat, 2024] y CLIPS [NASA, 2023] utilizan este paradigma.
- **Encadenamiento hacia atrás (Backward Chaining / Goal-Driven):** Parte de una meta u objetivo hipotético y trabaja hacia atrás, verificando qué reglas y hechos serían necesarios para demostrarlo. Es útil para problemas de diagnóstico y consulta, donde el espacio de búsqueda es grande pero el objetivo está bien definido. El lenguaje Prolog está basado en este paradigma.
- **Motores Deterministas (Policy Engines):** A menudo utilizados en contextos empresariales, se centran en evaluar un conjunto de condiciones contra un input concreto para producir una decisión única (Por ejemplo: aprobar o rechazar, calcular una tarifa). Suelen utilizar DSLs amigables y su flujo es más predecible y secuencial que el de los motores de inferencia pura [Taylor y Raden, 2007].

Es crucial distinguir entre un **BRE** y un **Motor de Flujo de Trabajo (Workflow Engine)**. Mientras que un BRE se centra en la *toma de decisiones* automática basada en condiciones y datos ("¿Qué debe pasar?"), un Workflow Engine se enfoca en la *orquestración de tareas* y procesos, gestionando el flujo entre actividades, asignando trabajo a personas o sistemas y controlando los plazos ("¿Quién debe hacer qué y cuándo?"). Mientras que un BRE produce conocimiento o una decisión; un Motor de Flujo de Trabajo ejecuta y coordina el trabajo de negocio. Ambos pueden ser complementarios en una arquitectura [Van Der Aalst et al., 2005].

Estado del arte

Esta sección revisa las plataformas de BRE más relevantes en el panorama actual y su aplicación específica en el dominio de la planificación o *scheduling*, con especial énfasis en el contexto académico para el cual se diseña la solución.

4.1. Plataformas de BRE Principales

El ecosistema de BRE ofrece soluciones que van desde motores de código abierto altamente técnicos hasta plataformas comerciales de bajo código (*low-code*) orientadas a usuarios de negocio. La elección depende de factores como la complejidad de las reglas, la integración con el stack tecnológico existente, el presupuesto y las habilidades del equipo.

4.2. Aplicación de BRE en Sistemas de Scheduling

La aplicación de BRE en el ámbito de la planificación (*scheduling*) es un caso de uso clásico y bien establecido que demuestra el valor tangible de esta tecnología. Empresas especializadas como **Celayix** sitúan explícitamente un motor basado en reglas en el núcleo de su solución de planificación de turnos, permitiendo a sus clientes codificar políticas complejas de personal, cumplimiento legal y preferencias operativas.

Los beneficios directos de integrar un BRE en un sistema de scheduling son múltiples:

- **Consistencia y Cumplimiento Automatizado:** El BRE aplica de manera uniforme y sin errores todas las políticas de negocio en cada planificación generada, asegurando el cumplimiento de normas laborales (ej., límites de horas, descansos obligatorios), contratos colectivos y regulaciones de seguridad (ej., cualificaciones requeridas para una tarea) [Charfi y Mezini, 2007].
- **Agilidad Operativa y Reducción de Costos:** Los gestores o administradores pueden modificar las reglas del negocio (ej., una nueva política de cualificación, cambios en la disponibilidad de recursos) en horas o días, sin depender de ciclos de desarrollo de software que pueden tomar semanas o meses. Esto reduce drásticamente el tiempo de adaptación y los costos asociados [Vanthienen y Caron, 2013].
- **Transparencia y Auditoría:** Toda decisión tomada por el sistema (ej., por qué se asignó un docente a una sala específica) puede ser explicada trazando las reglas que se activaron, lo que facilita la auditoría interna, la resolución de disputas y la mejora continua de los procesos [Li et al., 2020].

Tabla 1: Comparativa de Principales Plataformas de Business Rule Engines

Plataforma	Tipo	Lenguaje / Enfoque	Ventajas Principales	Consideraciones
Drools (Red Hat)	Código abierto	Java. Motor de reglas avanzado.	Potente, altamente personalizable y escalable. Amplia comunidad y documentación [Red Hat, 2024]. Soporta notación DMN [Object Management Group, 2018].	Curva de aprendizaje pronunciada. Requiere conocimientos técnicos profundos (Java). La gestión de reglas complejas puede volverse técnica.
InRule	Comercial	.NET. Enfoque en usabilidad y <i>low-code</i> .	Interfaz gráfica de autoría de reglas muy intuitiva. Excelente soporte y documentación. Buen rendimiento.	Costos de licencia pueden ser significativos. Menor flexibilidad para personalizaciones muy específicas frente a código abierto.
DecisionRules	Comercial (SaaS/Cloud)	<i>Low-code/No-code</i> , diseño API-first.	Muy accesible para usuarios de negocio. Gran flexibilidad de despliegue (cloud, on-premise). Fácil integración vía API REST.	Solución comercial con costos de suscripción asociados. La personalización profunda está limitada a las funcionalidades de la plataforma.
Salesforce B.R.E.	Comercial (Ecosistema)	<i>Low-code</i> , nativo en Salesforce.	Ideal para organizaciones ya invertidas en el ecosistema Salesforce. Muy intuitivo para analistas de negocio.	Fuertemente acoplado a la plataforma Salesforce. No es una solución independiente para sistemas externos.

4.3. Casos de Estudio Específicos para Scheduling Académico

En el contexto específico de la planificación académica (asignación horario-sala-docente), un BRE puede gestionar de forma declarativa un amplio espectro de restricciones complejas e interdependientes:

1. **Reglas de Calificación y Competencia:** Asegurar que un docente asignado a un curso posea las titulaciones, certificaciones o experiencia específica requerida para esa materia o nivel educativo [Alvarez *et al.*, 2018].
2. **Reglas de Disponibilidad y Conflictos:** Respetar la disponibilidad declarada de los docentes (horas de dedicación, ausencias programadas) y la ocupación previa de las salas, evitando superposiciones o dobles asignaciones [Muller y Rudova, 2005].
3. **Reglas de Carga Lectiva y Distribución Equitativa:** Controlar y distribuir equitativamente la cantidad de horas lectivas, créditos o número de cursos asignados a cada docente por semestre o año académico, considerando distintos tipos de contrato [Zhao *et al.*, 2019].
4. **Reglas de Preferencia y Prioridad:** Incorporar criterios de preferencia personal, antigüedad o desempeño en la asignación de horarios, tipos de curso (teórico vs. laboratorio) o sedes, de acuerdo con la política institucional [Chen *et al.*, 2022].
5. **Restricciones de Infraestructura y Pedagogía:** Asignar salas con capacidad, equipamiento (ej.: laboratorio de computación, proyector, mesas de trabajo) y características (ej.: anfiteatro) acordes a las necesidades pedagógicas del curso y al tamaño del grupo matriculado [Garcia *et al.*, 2021].

La integración propuesta de un BRE con el algoritmo heurístico de planificación existente en uPlanner sigue un modelo híbrido. El algoritmo se encarga de la búsqueda eficiente en el vasto espacio de soluciones, explorando combinaciones de asignaciones [Abdullah y Turabieh, 2012]. Por su parte, el BRE actúa como un validador en tiempo real, evaluando la viabilidad y calidad de cada asignación potencial contra el conjunto completo de reglas institucionales. Esta separación de responsabilidades permite **desacoplar** la lógica de negocio, volátil y compleja, del núcleo algorítmico, que es estable pero computacionalmente intensivo [Lewis *et al.*, 2016]. El resultado es un sistema más mantenible, comprensible y adaptable, donde los cambios en las políticas académicas no requieren modificar el código del algoritmo, y donde cada planificación generada viene acompañada de una justificación clara basada en las reglas aplicadas.

CAPÍTULO 5

PROPUESTA DE SOLUCIÓN

La propuesta de solución consiste en el diseño e implementación de un prototipo funcional de Motor de Reglas de Negocio (BRE), desarrollado íntegramente en Python, que actúa como capa de validación desacoplada respecto del algoritmo heurístico de planificación de uPlanner. El sistema permite definir, gestionar y evaluar restricciones académicas de forma declarativa, mediante un Lenguaje Específico de Dominio (DSL) basado en JSON, sin necesidad de modificar el código del motor de evaluación cada vez que cambia una política institucional.

5.1. Visión General de la Solución

La solución adopta un *enfoque híbrido* que integra el algoritmo heurístico existente de uPlanner con un BRE diseñado a medida. En este esquema, el algoritmo heurístico conserva su responsabilidad de explorar eficientemente el vasto espacio de soluciones de planificación [Abdullah y Turabieh, 2012], mientras que el motor de reglas asume el rol de *validador de factibilidad* para cada propuesta de asignación generada.

El principio arquitectónico fundamental de esta propuesta es la **separación de responsabilidades**: la lógica de negocio volátil y específica de cada institución educativa queda encapsulada en el archivo de definición de reglas (`rules.json`), mientras que el núcleo evaluador permanece estable y agnóstico al dominio. Este desacoplamiento, sustentado en principios de diseño de lenguajes específicos de dominio [Fowler, 2010], permite que expertos académicos incorporen, modifiquen o desactiven restricciones sin intervención del equipo de desarrollo. Así, se puede reducir drásticamente el tiempo de respuesta ante cambios institucionales [Vanthienen y Caron, 2013].

La Figura 3 ilustra los tres componentes principales de la solución y sus interacciones: el módulo de carga de datos, el motor de reglas y las interfaces de consulta.

5.2. Evolución del Modelo de Datos

El diseño de la solución comienza con la formalización del modelo de datos que representa el dominio de planificación académica. Esta etapa involucró una evolución desde un modelo conceptual inicial hacia un modelo más preciso, orientado específicamente a la evaluación de restricciones.

La Figura 14 (véase Anexo) muestra el modelo de datos original, que identifica las entidades fundamentales del dominio y sus relaciones, sirviendo como punto de partida para el análisis

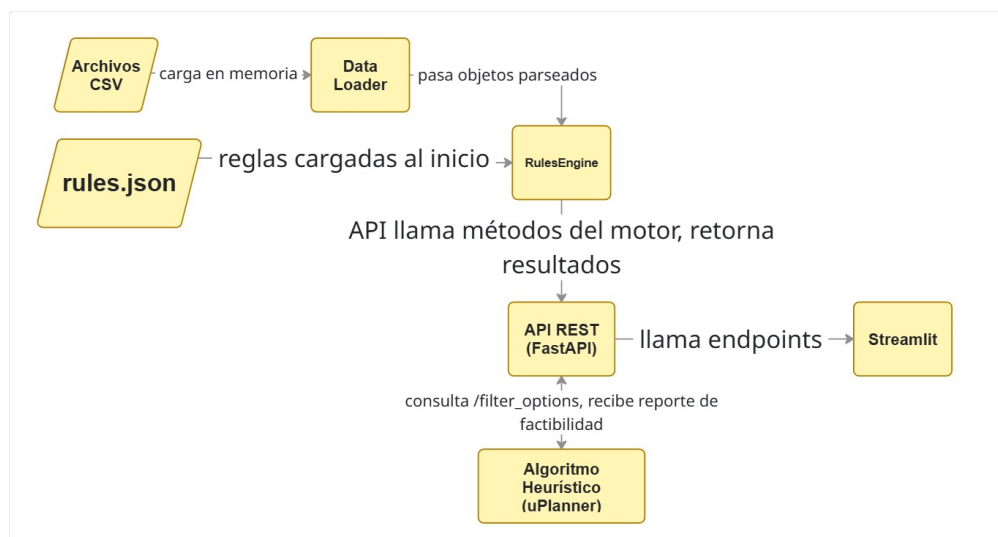


Figura 3: Arquitectura general de la solución propuesta. Elaboración propia.

de los atributos requeridos por cada restricción académica.

A partir de este análisis, se refinaron los atributos relevantes para la evaluación de reglas, priorizando aquellos que participan activamente en las condiciones de las restricciones y eliminando campos que no son consultados por ninguna regla del catálogo. El modelo resultante, presentado en la Figura 15 (véase Anexo), fue implementado utilizando el framework de validación Pydantic, que garantiza integridad de los datos en tiempo de ejecución y facilita la serialización de objetos complejos.

El modelo introduce cuatro entidades base que representan las dimensiones fundamentales de una asignación académica. En el eje temporal, `module` define un bloque horario concreto (identificador, número de módulo y horas de inicio y fin), mientras que `calendar` representa una instancia de tiempo asignable que combina una fecha de calendario, el día de la semana, la semana del año y el módulo correspondiente. En el eje físico, `infrastructure` describe un recurso de sala (presencial o virtual) con sus atributos de capacidad, tipo de uso y campus, e `infrastructure` y `scheduleRange` encapsulan respectivamente el recurso físico y el rango de calendarios válidos para una sección. El modelo se completa con `teacher`, que modela un docente con sus disponibilidades y restricciones contractuales, y con las entidades de configuración `groupScheduleOutput`, `presetSchedule` y `academicProgramming`, que capturan los grupos de estudiantes, las asignaciones manuales previas y la programación académica vigente. Las dos entidades centrales del motor, `Section` y `schedules`, se construyen sobre todas las anteriores y se describen en detalle al final del listado.

El modelo de implementación se organiza en torno a diez clases interrelacionadas que representan las entidades del dominio:

- `module`: Define un bloque horario con su identificador, número de módulo y horas de

inicio y fin.

- **calendar**: Representa una instancia de tiempo asignable, identificada por su fecha de calendario (`calendar_date`), día de la semana, semana del año y módulo asociado.
- **scheduleRange**: Encapsula un rango de calendarios válidos para la asignación de una sección.
- **infrastructure**: Describe un recurso físico o virtual, identificado por su sala (`id_classroom`), con atributos de capacidad, virtualidad, tipo de uso, campus de pertenencia y módulos de reserva pre y post asignación.
- **teacher**: Modela un docente con sus disponibilidades declaradas, cursos habilitados y restricciones contractuales: máximos de módulos diarios, semanales, consecutivos, de pausa diaria y de pausa semanal; máximos de secciones diarias, semanales y de cursos semanales; mínimos de módulos semanales, secciones semanales y cursos semanales; y número máximo de campus por día y por semana.
- **groupScheduleOutput**: Representa una agrupación de estudiantes compatibles en base a su plan de estudios, utilizado para verificar que no existan conflictos de horario entre secciones.
- **presetSchedule**: Codifica una asignación manual, con docente, sala y/o horario.
- **academicProgramming**: Registra la programación académica vigente para una sección (sala y docente habituales de clase).
- **Section**: Entidad central del modelo de datos. En el dominio original de uPlanner esta entidad corresponde a un *examen*, sin embargo fue renombrada a *Section* para reflejar con mayor claridad su rol como unidad primaria de asignación del motor. Agrega los grupos de estudiantes asociados, las reglas especiales aplicables, el rango de horario habilitado, y los requerimientos de sala, docente y modalidad (presencial o virtual). Incluye además la bandera `is_assigned_by_user`, que indica si la asignación fue establecida manualmente por un usuario del sistema.
- **schedules**: Objeto de propuesta de asignación que el motor evalúa. Contiene la sección, el rango de horario, la sala asignada, los calendarios seleccionados y el docente.

La clase `schedules` constituye el *input* primario del motor de reglas, es decir, cada instancia representa una propuesta concreta de asignación que debe ser validada contra el conjunto de restricciones académicas vigentes.

5.3. Carga de Datos desde Archivos CSV

El sistema incorpora un módulo de carga de datos implementado en la clase Exportador, cuya responsabilidad es leer los archivos CSV exportados por uPlanner y construir en memoria las instancias Pydantic del modelo de datos descritas en la sección anterior. La función `iterar_cargas()` actúa como orquestador, invocando los diez métodos de carga en un orden determinado por las dependencias entre entidades, de modo que cada modelo disponga de las referencias a los objetos previamente construidos al momento de ser instanciado.

El proceso de carga sigue la secuencia indicada en la Tabla 2, respetando el orden de dependencias del modelo: las entidades base se cargan primero y las entidades compuestas se construyen a partir de las referencias ya disponibles en memoria.

Tabla 2: Secuencia de carga de datos desde archivos CSV.

Orden	Entidad	Archivos CSV de origen
1	groupScheduleOutput	groupScheduleOutput.csv
2	infrastructure	infrastructure.csv, parameters.csv
3	module	modulesDefinition.csv
4	calendar	scheduleRange.csv, presetCalendar.csv
5	scheduleRange	scheduleRange.csv
6	teacher	teacherAvailableModules.csv, teacherContract.csv, teacherCourseActivity.csv
7	presetSchedule	presetRestriction.csv
8	academicProgramming	academicProgrammingSchedule.csv
9	Section	sectionRequirements.csv, sectionAssignmentRules.csv, sectionUsageType.csv
10	schedules	schedules.csv

Varias entidades requieren transformaciones no triviales durante la carga. La entidad `calendar` se construye a partir de la unión de los archivos `scheduleRange.csv` y `presetCalendar.csv`: ambos conjuntos se concatenan, se eliminan duplicados por combinación de atributos temporales, y se genera un identificador sintético `id_calendar` secuencial. Este identificador sintético actúa como clave de cruce para las entidades que referencian calendarios, como `scheduleRange`, `teacher` y `presetSchedule`. La entidad `teacher` combina tres archivos: disponibilidad declarada de módulos, términos del contrato y cursos habilitados, que se cruzan por `id_teacher`. La entidad `Section` es la más compleja, pues consolida requerimientos base, reglas de asignación especiales y tipos de uso desde tres archivos independientes, y resuelve además las referencias a `groupScheduleOutput`, `presetSchedule` y `academicProgramming` mediante mapeos auxiliares construidos duran-

te las etapas anteriores.

Cada error producido durante la carga de cualquier entidad es capturado de forma independiente y registrado en una lista de errores que se retorna junto con los datos cargados, permitiendo una carga parcial tolerante a fallos en archivos individuales.

5.4. Lenguaje de Definición de Reglas Declarativo

Uno de los aportes centrales de la presente memoria es el diseño de un DSL basado en JSON que permite expresar restricciones académicas de forma legible, estructurada y modificable sin conocimientos de programación [Fowler, 2010]. Este lenguaje define la gramática mediante la cual los administradores del sistema pueden codificar cualquier restricción del dominio en el archivo `rules.json`.

5.4.1. Estructura de una Regla

Cada regla en el catálogo es un objeto JSON con los siguientes atributos:

- **rule_id**: Identificador único de la regla.
- **name**: Nombre descriptivo legible.
- **description**: Descripción en lenguaje natural del propósito de la restricción.
- **enabled**: Booleano que permite activar o desactivar la regla sin eliminarla del catálogo.
- **error_message**: Mensaje explicativo retornado cuando la regla no se cumple, orientado a la auditoría y corrección.
- **scope**: Alcance de evaluación; puede ser "section" (evaluación individual) o "group" (evaluación colectiva).
- **group_by**: Lista de variables de modelo utilizadas para agrupar propuestas en las reglas de alcance grupal.
- **applicability_conditions** (opcional): Condiciones previas que determinan si la regla aplica a una propuesta concreta. Si no se satisfacen, la regla se considera no aplicable y su resultado es válido por omisión.
- **conditions**: Bloque de condiciones que, al satisfacerse todas ("all") o al menos una ("any"), determinan que la asignación es válida.

Cada condición dentro de un bloque `conditions` o `applicability_conditions` se compone de tres elementos:

- **fact:** Ruta al atributo del objeto `schedules` que se desea evaluar, expresada en notación de puntos. Por ejemplo, `id_section.id_teacher.nm_maximum_weekly_modules` navega desde la propuesta hasta el atributo de módulos semanales máximos del docente asignado a la sección. Cuando algún segmento de la ruta resuelve a una lista, el motor extrae el atributo de cada elemento de forma automática.
- **operator:** Identificador de la función de evaluación que se aplicará sobre los valores resueltos. Determina el tipo de comparación (igualdad, orden, pertenencia a conjunto, cardinalidad, etc.). El conjunto completo de operadores disponibles se detalla en la Sección 5.6.
- **value:** Valor de referencia contra el cual se evalúa el hecho. Puede ser un literal (número, cadena o booleano) o una referencia dinámica a otro atributo del objeto, expresada como `{"fact": ruta.al.atributo"}`. En este último caso, el motor resuelve el valor en tiempo de evaluación sobre la misma propuesta, lo que permite comparar dos atributos del modelo entre sí sin codificar valores fijos en la regla.

El siguiente es un ejemplo de una regla de alcance grupal que verifica que todas las secciones de un mismo curso compartan el mismo horario de asignación (Regla R001):

```
{
  "rule_id": "R001",
  "name": "Asignar cursos en mismo horario",
  "scope": "group",
  "group_by": ["id_section.id_course"],
  "enabled": true,
  "conditions": {
    "all": [
      {
        "fact": "id_calendar",
        "operator": "allEqual",
        "value": { "fact": "id_calendar" }
      }
    ]
  }
}
```

5.4.2. Resolución de Hechos mediante Notación de Puntos

El sistema de resolución de hechos permite acceder a los atributos anidados desde el objeto `schedules` mediante una notación de puntos (*dot notation*). Por ejemplo, la ruta `"id_section.id_teacher.nm_maximum_weekly_modules"` navega desde la propuesta de asignación hasta el atributo de módulos semanales máximos del docente de la sección evaluada.

Cuando un segmento de la ruta resuelve a una lista de objetos, el motor aplana automáticamente la estructura, extrayendo el atributo solicitado de cada elemento. Este mecanismo permite que una misma sintaxis de condición opere correctamente sobre relaciones uno-a-uno y uno-a-muchos sin requerir sintaxis adicional, simplificando significativamente la definición de reglas que involucran relaciones complejas del modelo.

5.4.3. Alcance de Evaluación: Sección y Grupo

El motor distingue dos alcances de evaluación que determinan cómo se procesan las condiciones de una regla:

- **Alcance individual** (`scope: "section"`): La regla se evalúa de forma independiente para cada propuesta de asignación. Los hechos se resuelven sobre el objeto `schedules` individual. Este alcance es adecuado para restricciones que dependen únicamente de los atributos de la sección asignada, como la capacidad de la sala (R008), el cumplimiento del tipo de recurso (R009) o la virtualidad del aula (R018).
- **Alcance de grupo** (`scope: "group"`): La regla se evalúa en el contexto de todas las propuestas que comparten una misma clave de agrupación, definida por los campos `group_by`. El motor reúne las propuestas del grupo y evalúa el operador sobre el conjunto de valores de sus miembros. Este alcance es adecuado para restricciones que involucran múltiples secciones simultáneamente, como la detección de topes de horario (R012, R024), la sincronización de calendarios entre secciones de un mismo curso (R001), o el control de la carga semanal de un docente (R016W).

En las reglas de grupo, el campo `group_by` admite rutas de hechos que resuelven a listas. Cuando un campo de agrupación tiene múltiples valores para una misma propuesta, el motor genera todas las combinaciones posibles mediante producto cartesiano, asegurando que la propuesta sea evaluada en todos los grupos pertinentes.

Para los operadores de promedio semanal (`averageCountPerWeekLe/Lt/Gt/Ge`), se aplica una estrategia adicional de deduplicación: en lugar de calcular un promedio independiente por cada clave de grupo, el motor reúne de forma única los miembros de *todas* las claves

pertinentes de la propuesta antes de calcular el promedio. Esto garantiza que una propuesta que pertenece a múltiples grupos simultáneos (por ejemplo, por estar programada en varios días o pertenecer a varios grupos de estudiantes) produzca un único valor de promedio consistente, evitando que el mismo examen falle en todas las sub-reglas de distribución simultáneamente por haber obtenido promedios distintos para distintas claves.

5.4.4. Condiciones de Aplicabilidad

Las condiciones de aplicabilidad (`applicability_conditions`) permiten delimitar el universo de propuestas para las cuales una regla es relevante, sin penalizar aquellas a las que no corresponde aplicar. Por ejemplo, la regla R013 ("Asignar sin supervisor") solo es pertinente para las secciones cuyo campo `id_rule` contenga el código `.^ASSIGN_NO_INSTRUCTOR`; para el resto, la regla se omite y su resultado se registra como válido.

Este mecanismo evita la proliferación de condiciones negativas en el bloque `conditions` de la regla, haciendo que cada regla describa exclusivamente el comportamiento esperado para su caso de uso específico. El resultado es un conjunto de reglas más legible y mantenible, donde la intención de cada restricción es inmediatamente comprensible [Ross, 2013].

5.5. Arquitectura e Implementación del Motor de Reglas

El motor de reglas está implementado en la clase `RulesEngine`, que encapsula la carga de reglas desde JSON, la resolución de hechos mediante notación de puntos, la evaluación de condiciones y la gestión del estado necesario para las evaluaciones grupales.

La clase expone tres métodos principales de evaluación:

- `evaluate()`: Evalúa una sola propuesta de asignación contra el conjunto de reglas habilitadas o un subconjunto especificado por nombre. Recibe opcionalmente el contexto completo de propuestas para habilitar la evaluación de reglas grupales.
- `evaluate_all()`: Evalúa el conjunto completo de propuestas cargadas en una sola invocación. Construye el índice de grupos una única vez y lo reutiliza para todas las evaluaciones, siendo significativamente más eficiente que invocar `evaluate()` individualmente para cada propuesta.
- `filter_options()`: Recibe una lista de opciones candidatas (salas, docentes o calendarios) e itera sobre ellas, insertando cada opción en el campo correspondiente de una propuesta base y evaluando las reglas aplicables. Retorna un informe detallado de validez por opción y por regla, que puede ser consultado directamente por el algoritmo heurístico para identificar asignaciones factibles [Li et al., 2020].

El ciclo de evaluación para una regla individual sigue los pasos representados en la Figura 4. Primero se verifica si la regla está habilitada, luego se evalúan las condiciones de aplicabilidad (si existen), y finalmente se procesan las condiciones principales utilizando el operador declarado. Cada resultado incluye un campo `validity_type` que distingue tres estados posibles: (1) `"valid"` (condiciones satisfechas), (2) `"not_applicable"` (condiciones de aplicabilidad no cumplidas; la regla no corresponde a esta propuesta y se omite sin penalizar) e (3) `"invalid"` (condiciones incumplidas). En este ultimo caso se muestra el mensaje de error correspondiente.

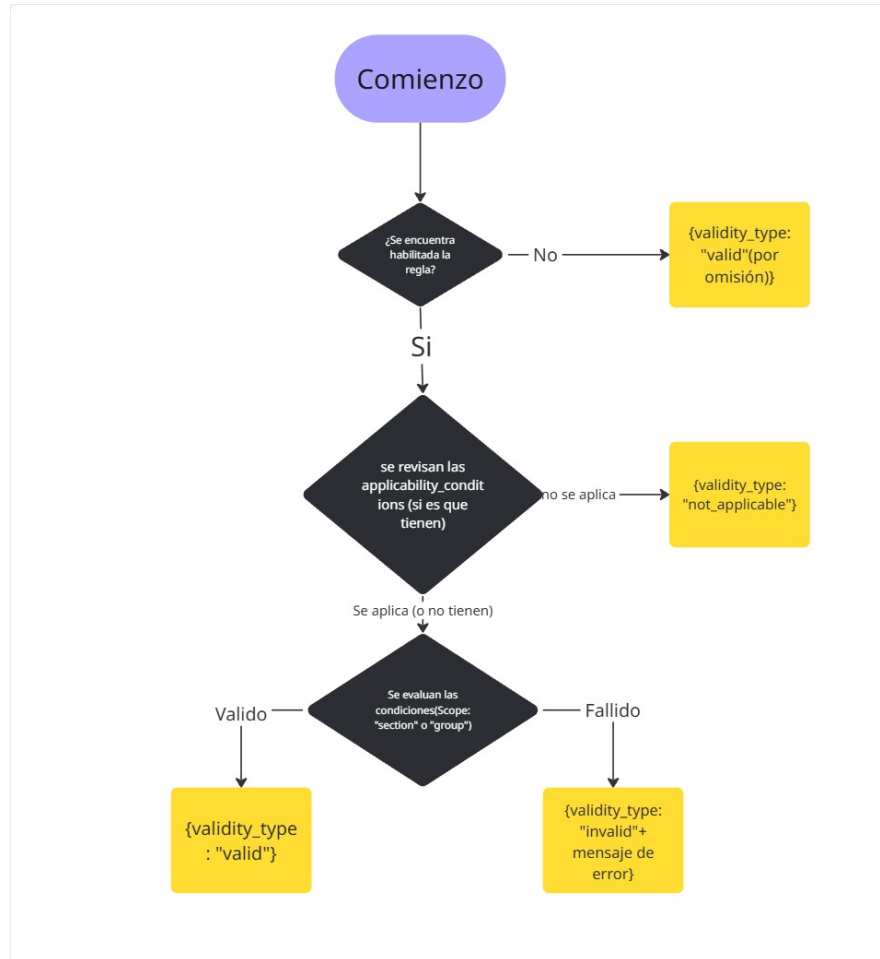


Figura 4: Ciclo de evaluación de una regla individual en el motor. Elaboración propia.

5.6. Diseño del Conjunto Mínimo de Operadores

Un objetivo de diseño explícito del motor fue **minimizar el número de operadores** requeridos para expresar el conjunto completo de restricciones académicas del dominio. Un conjunto compacto de operadores reduce la complejidad del código del motor, facilita la in-

corporación de nuevos operadores en el futuro y limita la superficie de posibles errores de implementación [Giarratano y Riley, 2005].

La clave que habilita este principio es la **interfaz unificada de operadores**: todos los operadores reciben como primer argumento una *lista* de valores resueltos (`values: List[Any]`), independientemente de si la evaluación es individual (lista con un elemento) o grupal (lista con N elementos). Lo anterior, elimina la necesidad de definir operadores unitarios y grupales separados para la mayoría de las comparaciones, reduciendo la duplicación de lógica.

El conjunto resultante comprende **18 operadores**, organizados en cinco categorías funcionales, tal como se detalla en la Tabla 3.

Los operadores de la categoría *Escalar/Grupo* son los que más evidencian el beneficio del diseño de interfaz unificada dado que la misma función implementa tanto la verificación sobre un valor individual como la verificación colectiva sobre todos los miembros de un grupo, requiriendo en ambos casos una sola línea de lógica.

5.7. Optimizaciones de Rendimiento

La evaluación de reglas sobre el conjunto completo de propuestas de asignación presenta un desafío de rendimiento relevante, dado que las reglas de alcance grupal requieren acceder a los miembros del grupo para cada propuesta evaluada. En caso de no haber optimización, lo anterior implica una búsqueda lineal en el conjunto de propuestas por cada condición grupal, resultando en una complejidad cuadrática respecto al tamaño del conjunto.

Para mitigar este costo, el motor implementa dos mecanismos complementarios de caché que se construyen una única vez al inicio de cada evaluación en lote:

- **Caché de volcado (*dump cache*)**: Al iniciar una evaluación en lote, el motor serializa cada objeto `schedules` en un diccionario Python mediante `model_dump()` y almacena el resultado en caché, indexado por el identificador de objeto en memoria. Esto evita la re-serIALIZACIÓN repetida del objeto en cada condición que accede a sus atributos durante la evaluación.
- **Índice de grupos (*group index*)**: Antes de la primera evaluación grupal, el motor construye un índice que asocia, para cada combinación de campos `group_by` utilizada por alguna regla del catálogo, los identificadores de las propuestas que comparten cada clave de grupo. Durante la evaluación, la obtención de los miembros de un grupo se reduce a una consulta en tiempo $O(1)$ sobre este índice, en lugar de un recorrido lineal sobre el conjunto completo. El índice se reconstruye únicamente cuando el conjunto de propuestas cambia.

La combinación de estos mecanismos hace que la evaluación en lote mediante

Tabla 3: Conjunto de operadores implementados en el motor de reglas.

Operador	Categoría	Descripción
allEqual	Comparación	Todos los valores de los miembros del grupo son iguales al objetivo.
notEqual	Comparación	No todos los valores son iguales al objetivo.
greaterThan	Comparación	Todos los valores son estrictamente mayores al objetivo.
lessThan	Comparación	Todos los valores son estrictamente menores al objetivo.
greaterThanOrEqual	Comparación	Todos los valores son mayores o iguales al objetivo.
lessThanOrEqual	Comparación	Todos los valores son menores o iguales al objetivo.
countEqual	Conteo	La cantidad de miembros del grupo es exactamente igual al objetivo.
countLessThanOrEqual	Conteo	La cantidad de miembros del grupo es menor o igual al objetivo.
countGreaterThanOrEqual	Conteo	La cantidad de miembros del grupo es mayor o igual al objetivo.
setContains	Conjunto	Todos los valores de la lista contienen al elemento objetivo.
hasSize	Colección	La colección resuelta tiene exactamente N elementos.
hasConsecutiveBlocks	Colección	Los calendarios asignados incluyen al menos N bloques consecutivos en el mismo día.
groupIsDisjointWith	Conflicto	Ningún par de secciones del grupo comparte una misma franja horaria.
groupLessThanOrEqual	Agregación	La suma total de módulos del grupo no supera el límite definido.
averageCountPerWeekLe	Promedio semanal	El promedio de secciones del grupo por semana hábil ¹ es menor o igual al objetivo.
averageCountPerWeekLt	Promedio semanal	El promedio de secciones del grupo por semana hábil es estrictamente menor al objetivo.
averageCountPerWeekGt	Promedio semanal	El promedio de secciones del grupo por semana hábil es estrictamente mayor al objetivo.
averageCountPerWeekGe	Promedio semanal	El promedio de secciones del grupo por semana hábil es mayor o igual al objetivo.

`evaluate_all()` sea órdenes de magnitud más rápida que invocar `evaluate()` individualmente para cada propuesta, lo que resulta crítico para su integración dentro de algoritmos de búsqueda heurística de alta frecuencia de consulta [Muller y Rudova, 2005].

5.8. Interfaces de la Solución

La solución incluye dos interfaces complementarias que facilitan la validación e inspección del motor de reglas: (1) una API REST para integración programática con el algoritmo heurístico y (2) una interfaz web para exploración interactiva por parte del equipo de desarrollo y administradores académicos.

5.8.1. API REST con FastAPI

La API REST, implementada con FastAPI, expone el motor de reglas como un servicio web independiente con siete endpoints (resumidos en la Tabla 4). El endpoint `/filter_options` es particularmente relevante para la integración con el algoritmo heurístico: permite consultar, para una sección parcialmente resuelta, qué opciones de sala, docente o calendario son compatibles con las reglas vigentes antes de proceder con la asignación, convirtiendo al motor en un oráculo de factibilidad que guía las decisiones del algoritmo en tiempo real [Li *et al.*, 2020].

Tabla 4: Endpoints de la API REST del motor de reglas.

Endpoint	Método	Descripción
<code>/load_csv</code>	POST	Carga datos de entrada desde un directorio de archivos CSV y los almacena en memoria.
<code>/evaluate</code>	POST	Evalúa una propuesta de asignación individual contra las reglas activas o un subconjunto.
<code>/evaluate_batch</code>	POST	Evalúa en lote todas las propuestas cargadas, con índice de grupos preconstruido.
<code>/filter_options</code>	POST	Filtra una lista de opciones (salas, docentes, calendarios) y retorna un reporte de validez por opción y regla.
<code>/rules</code>	GET	Retorna el catálogo de reglas actualmente cargadas en el motor.
<code>/schedules</code>	GET	Retorna las propuestas de asignación cargadas desde CSV.
<code>/sections</code>	GET	Retorna las secciones cargadas desde CSV.

5.8.2. Interfaz Web con Streamlit

La interfaz web, implementada con Streamlit, ofrece cinco páginas funcionales orientadas tanto a usuarios no técnicos como al equipo de desarrollo:

- **Crear Regla:** Formulario dinámico para definir nuevas reglas y persistirlas directamente en `rules.json`, permitiendo agregar condiciones múltiples y seleccionar operadores desde un menú desplegable.
- **Probar Reglas:** Permite seleccionar una propuesta cargada y evaluarla contra reglas activas o un subconjunto, visualizando el resultado de validez por cada regla junto al mensaje de error cuando corresponde.
- **Listar Reglas:** Muestra el catálogo de reglas con sus condiciones expandibles en formato tabla, facilitando la auditoría del conjunto de restricciones vigentes.
- **Probar Filtro:** Interfaz para la funcionalidad `/filter_options`, que permite explorar interactivamente qué opciones son válidas para una sección dada y por qué las inválidas fallan.
- **Resumen de Reglas:** Vista de evaluación en lote con métricas agregadas y gráfico de barras apilado que desglosa, por cada regla, la cantidad de horarios en estado válido, parcialmente válido (N/A) e inválido. Incluye paginación, filtros por regla y una tabla de totales global, siendo la herramienta principal para identificar las restricciones con mayor tasa de violación en el conjunto de asignaciones vigente.

Las Figuras 5 a 13 muestran capturas de pantalla de las cinco páginas de la interfaz Streamlit.

1. Define las Condiciones

Añade y configura una o más condiciones que debe cumplir la regla.

Fact (Atributo): sección.cursoid Operador: equal ☐ Literal ☒ Fact Valor (otro Fact): recurso.capacidad [X]

Añadir Condición

2. Guarda la Regla

Define los detalles de la regla y haz clic en guardar.

ID de la Regla: Ej: R008 Descripción de la Regla: []

Nombre de la Regla: Ej: docente_no_trabaja_viernes Mensaje de Error si falla: []

Lógica de las condiciones: ☒ all ☐ any

☒ Habilitar esta regla por defecto

Guardar Nueva Regla

Figura 5: Página *Crear Regla*: formulario de dos pasos para definir condiciones y guardar una nueva regla en `rules.json`. Elaboración propia.

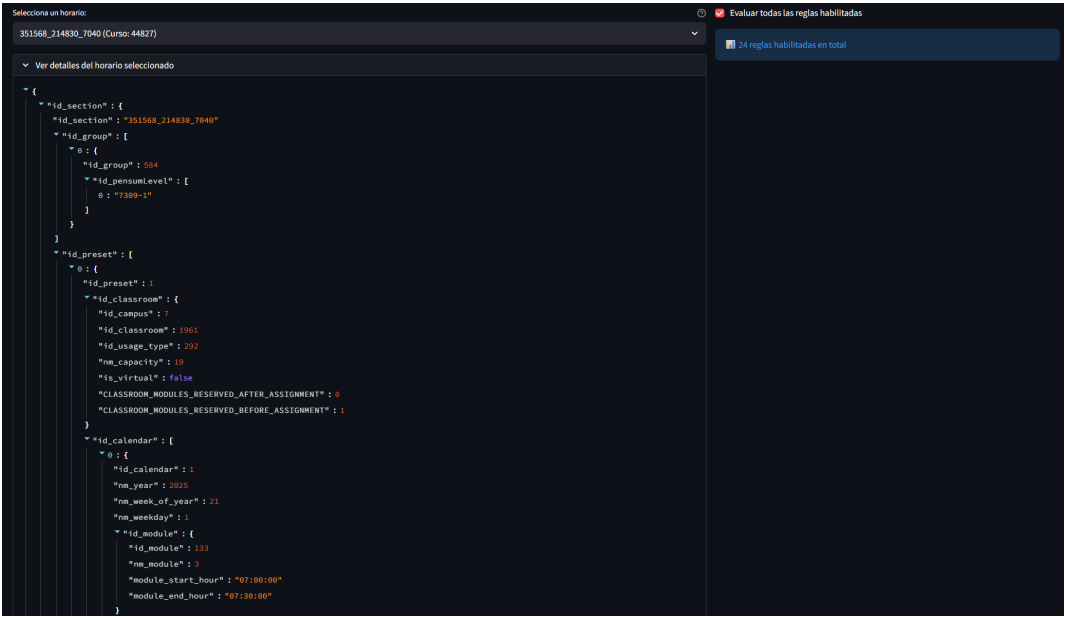


Figura 6: Página *Probar Reglas* (a): selector de propuesta y panel desplegable con los atributos del objeto `schedules` seleccionado. Elaboración propia.

Selección Horario

351568_214830_7040 (Curso: 44827)

Ver detalles del horario seleccionado

Selección Reglas

Evaluar todas las reglas habilitadas

24 reglas habilitadas en total

Evaluar Horario

Resultados de la Evaluación

Total de Reglas	Válidas	Inválidas
24	20	4

Mostrar reglas válidas

Mostrar reglas inválidas

Regla	Descripción	Estado	Mensaje de Error
Asignar cursos en mismo horario	La asignación debe ser igual para todas las secciones del mismo curso.	Inválida	Las secciones de la programación académica no co
Asignaciones manuales mantenidas por el algoritmo	Las asignaciones manuales deben mantenerse en el mismo recurso, docente y calendario.	Válida	None
No más de 3 exámenes por día por plan-nivel	No puede haber más de 3 exámenes programados por día para un mismo plan-nivel.	Válida	None
Asignación de Horario y sala presencial o virtual por requerimiento	La cantidad de bloques asignados debe coincidir con los bloques semanales del curso.	Válida	None
Asignación de supervisor(contador)	El recurso debe tener un supervisor.	Válida	None
Promedio de 1 o menos exámenes por semana por grupo	El grupo de estudiantes tiene en promedio 1 o menos exámenes por semana (semana = 6 días).	Válida	None
Promedio entre 1 y 3 exámenes por semana por grupo	El grupo de estudiantes tiene en promedio más de 1 y menos de 3 exámenes por semana (semana = 6 días).	Inválida	El promedio de exámenes por semana para este gr
Promedio de 3 o más exámenes por semana por grupo	El grupo de estudiantes tiene en promedio 3 o más exámenes por semana (semana = 6 días).	Inválida	El promedio de exámenes por semana para este gr
Cumple con la capacidad de la sala	El recurso debe tener capacidad para los estudiantes.	Válida	None
Cumple con tipo de recurso presencial configurado	Un recurso físico debe coincidir con el tipo de uso requerido por la sección.	Válida	None

Ver respuesta JSON completa

Figura 7: Página *Probar Reglas* (b): tabla de resultados de evaluación por regla, con indicador de estado y mensaje de error cuando corresponde. Elaboración propia.

Regla	Descripción	Estado	Mensaje de Error
Asignación dentro de las fechas del calendario	Una sección debe asignarse dentro del rango horario y fecha de calendario correspondiente.	Valida	None
Cantidad de módulos requeridos	Una sección no puede ser asignada a un calendario que no cuente con la cantidad de módulos semanales requeridos	Valida	None
Tope de horario por estudiante	Un grupo compatible no puede tener tope de horario	Valida	None
Regla de supervisor/Asignar sin supervisor	Se debe hacer una asignación sin supervisor.	Valida	La regla no es aplicable.
Asignar cualquier supervisor	Se asigna supervisor si la sección lo requiere.	Valida	None
Se cumplen las restricciones de contrato docente para máximo de módulos semanales	Se asigna docentes siempre y cuando se cumplan las restricciones semanales de sus contratos.	Valida	None
Se asigna supervisor según su disponibilidad	La asignación debe ser a una sala presencial si la sección es presencial.	Valida	None
Asignar aula virtual	La sección debe ser asignada a un aula virtual.	Valida	La regla no es aplicable.
Asignar recurso de la programación	Una sección presencial no puede ser asignada a un aula virtual.	Invalida	La sección es presencial, por lo que debe asignarse
Programar en último horario de clases	Las secciones con esta regla deben estar asignadas en el último bloque.	Valida	La regla no es aplicable.
Programar en primer horario de clases	Las secciones con esta regla deben estar asignadas en el primer bloque.	Valida	La regla no es aplicable.

Ver respuesta JSON completa

```
{
  "0": {
    "rule_name": "Asignar cursos en mismo horario"
    "description": "La asignación debe ser igual para todas las secciones del mismo curso."
    "is_valid": false
    "validity_type": "invalid"
    "error_message": "Las secciones de la programación académica no coinciden en el calendario asignado."
  },
  "1": {
    "rule_name": "Asignaciones manuales mantenidas por el algoritmo"
    "description": "Las asignaciones manuales deben mantenerse en el mismo recurso, docente y calendario."
    "is_valid": true
    "validity_type": "valid"
    "error_message": null
  },
  "2": {
    "rule_name": "No más de 3 exámenes por día por plan-nivel"
    "description": "No puede haber más de 3 exámenes programados por día para un mismo plan-nivel."
    "is_valid": true
    "validity_type": "valid"
    "error_message": null
  }
}
```

Figura 8: Página *Probar Reglas* (c): continuación de la tabla de resultados y respuesta JSON completa del motor de reglas. Elaboración propia.

Listado de Reglas Existentes
Aquí puedes ver todas las reglas que están actualmente configuradas en el motor.
Se encontraron 24 reglas.
Asignar cursos en mismo horario (ID: R001)
Asignaciones manuales mantenidas por el algoritmo (ID: R002)
No más de 3 exámenes por día por plan-nivel (ID: R003)
asignación de Horario y sala presencial o virtual por requerimiento (ID: R004)
asignación de supervisor(contador) (ID: R006)
Promedio de 1 o menos exámenes por semana por grupo (ID: R007_1)
Promedio entre 1 y 3 exámenes por semana por grupo (ID: R007_2)
Promedio de 3 o más exámenes por semana por grupo (ID: R007_3)
Cumple con la capacidad de la sala (ID: R008)
Cumple con tipo de recurso presencial configurado (ID: R009)
Asignación dentro de las fechas del calendario (ID: R010)
Cantidad de módulos requeridos (ID: R011)
Tope de horario por estudiante (ID: R012)

Figura 9: Página *Listar Reglas*: catálogo de las 24 reglas configuradas, con nombre e identificador, desplegables para auditar condiciones individuales. Elaboración propia.

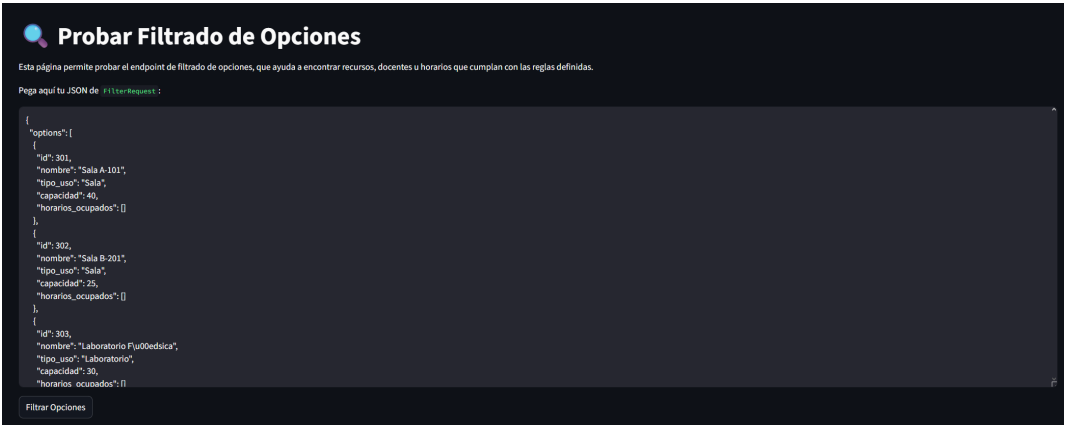


Figura 10: Página *Probar Filtrado de Opciones*: entrada JSON de opciones candidatas (salas, docentes o calendarios) y botón de filtrado que invoca `filter_options`. Elaboración propia.



Figura 11: Página *Resumen de Reglas* (a): métricas globales (4.557 horarios evaluados, promedio de 18,1/24 reglas válidas) y gráfico de barras apilado con distribución de estados por regla. Elaboración propia.

Filtros de Tabla

☐ Mostrar solo horarios con al menos un fallo

Columnas de reglas a mostrar (vacío = todas):
Choose options

Mostrando 4557 de 4557 horarios.

Anterior

Página 1 de 46

Siguiente

Horario	Válidos	R001	R002	R003	R004	R006	R007_1	R007_2	R007_3	R008	R009	R010	R011	R012	R013	R014	R016W	R017	R018	R019	R020	R021	R022	R023	R024
351588_214830_7040 (C-44827)	20/24	✗	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓
351569_214831_7040 (C-44827)	19/24	✗	✓	✗	✓	✓	✗	✗	✗	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
351575_214837_1541 (C-44837)	17/24	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
351576_214838_1541 (C-44837)	17/24	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
351594_214879_6936 (C-44903)	22/24	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
351595_214876_6936 (C-44903)	22/24	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
351614_214917_6829 (C-44925)	19/24	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
351615_214918_6829 (C-44925)	17/24	✗	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
351616_214919_6829 (C-44925)	17/24	✗	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
351617_214920_6829 (C-44925)	17/24	✗	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓ Válido | ✓ Parcialmente válido (N/A) — condiciones de aplicabilidad no cumplidas | ✗ Inválido

Anterior

Página 1 de 46

Siguiente

Figura 12: Página *Resumen de Reglas* (b): tabla paginada de horarios individuales con estado por regla (*válido*, *N/A* e *inválido*), filtrable y exportable a CSV. Elaboración propia.

Totales por regla (todos los horarios evaluados)

Regla	Válido / Total	Totalmente válidos	Parcialmente válidos (N/A)	Inválidos
R001 - Asignar cursos en mismo horario	2570/4557		2570	1987
R002 - Asignaciones manuales mantenidas por el algoritmo	4557/4557		4557	0
R003 - No más de 3 exámenes por día por plan-nivel	370/4557		370	4187
R004 - asignación de Horario y sala presencial o virtual por requerimiento	3801/4557		3801	756
R006 - asignación de supervisor(contador)	4557/4557		4557	0
R007_1 - Promedio de 1 o menos exámenes por semana por grupo	523/4557		523	4034
R007_2 - Promedio entre 1 y 3 exámenes por semana por grupo	652/4557		652	3905
R007_3 - Promedio de 3 o más exámenes por semana por grupo	3382/4557		3382	1175
R008 - Cumple con la capacidad de la sala	3032/4557		3032	1525
R009 - Cumple con tipo de recurso presencial configurado	2529/4557		2529	2028

Figura 13: Página *Resumen de Reglas* (c): tabla de totales por regla con conteo de válidos, N/A e inválidos sobre la totalidad de las 4.557 propuestas evaluadas. Elaboración propia.

5.9. Catálogo de Reglas Implementadas

El motor incluye un catálogo inicial de 24 reglas de negocio que cubren las principales restricciones del dominio de planificación académica de uPlanner [Alvarez et al., 2018]. La Tabla 5 presenta una descripción resumida de cada regla y su alcance de evaluación.

Tabla 5: Catálogo de reglas de negocio implementadas en el motor de reglas. Elaboración personal

ID	Nombre	Alcance	Descripción
R001	Asignar cursos en mismo ho-rario	Grupo	Todas las secciones del mismo curso de-ben compartir el mismo calendario.
R002	Asignaciones manuales man-tenidas	Sección	Las asignaciones preconfiguradas deben conservar su docente, sala y calendarios originales.

Tabla 5 – Continuación

ID	Nombre	Alcance	Descripción
R003	Máximo 3 exámenes por día por plan-nivel	Grupo	No puede haber más de 3 exámenes en un mismo día para un plan-nivel.
R004	Horario y sala según requerimiento	Sección	La virtualidad de la sala debe coincidir con la requerida por la sección.
R006	Asignación de supervisor	Sección	Toda sección debe tener un docente asignado.
R007_1	Distribución de exámenes por día	Grupo	Restricción sobre la distribución diaria de exámenes por estudiante (1, 2 o más).
R007_2	Máximo 2 exámenes al día en promedio	Grupo	El promedio de exámenes diarios por estudiante no supera 2.
R007_3	Máximo 3 exámenes al día en promedio	Grupo	El promedio de exámenes diarios por estudiante no supera 3.
R008	Capacidad de sala suficiente	Sección	La capacidad de la sala asignada debe ser mayor o igual al número de estudiantes de la sección.
R009	Tipo de recurso presencial	Sección	El tipo de uso del recurso físico debe coincidir con el requerido por la sección.
R010	Asignación dentro del rango del calendario	Sección	Los calendarios asignados deben pertenecer al rango de horario válido para la sección.
R011	Cantidad de módulos requeridos	Sección	El calendario asignado debe contener la cantidad de módulos semanales requeridos por la sección.
R012	Tope de horario por estudiante	Grupo	Un grupo de estudiantes compatibles no puede presentar tope de horario entre sus secciones.
R013	Asignar sin supervisor	Sección	Las secciones que así lo requieren no deben tener docente asignado.
R014	Asignar cualquier supervisor	Sección	Las secciones que lo requieren deben tener asignado un docente.
R016W	Restricciones semanales de contrato docente	Grupo	Los módulos semanales del docente no deben superar el máximo establecido en su contrato.
R017	Supervisor según disponibilidad	Sección	El calendario asignado debe estar dentro de la disponibilidad declarada del docente.
R018	Asignar aula virtual	Sección	Las secciones que lo requieren deben asignarse a un aula virtual.
R019	Asignar recurso de la programación	Sección	Las secciones con esta regla especial deben asignarse a la sala indicada en la programación académica.
R020	Programar en último bloque horario	Sección	La sección debe asignarse en el último bloque disponible del día.

Tabla 5 – Continuación

ID	Nombre	Alcance	Descripción
R021	Programar en primer bloque horario	Sección	La sección debe asignarse en el primer bloque disponible del día.
R022	Simultáneo en bloque 23:00-24:00	Grupo	Las secciones del bloque 23:00-24:00 deben asignarse de forma simultánea.
R023	Programar secciones en simultáneo	Grupo	Las secciones del mismo curso con esta regla deben asignarse en el mismo calendario.
R024	Tope de horario por sala	Grupo	No deben existir dos secciones asignadas a la misma sala en el mismo bloque horario.

Las reglas R013, R014, R018, R019, R020, R021, R022 y R023 incluyen condiciones de aplicabilidad que restringen su evaluación a las secciones que portan los códigos de regla especial correspondientes en su atributo `id_rule`, siguiendo el mecanismo descrito en la Sección 5.4.4. Las 16 reglas restantes se evalúan sobre la totalidad de las propuestas activas.

CAPÍTULO 6

VALIDACIÓN DE LA SOLUCIÓN

El presente capítulo tiene por objetivo demostrar que el prototipo de motor de reglas desarrollado constituye una solución funcional y aplicable al dominio de planificación académica de uPlanner. Para ello, se ejecutó una evaluación en lote sobre un conjunto de propuestas de asignación reales, analizando el comportamiento del motor frente a las 24 reglas del catálogo y verificando la coherencia de los resultados producidos.

6.1. Metodología de Validación

La validación se planteó como una **evaluación empírica sobre datos reales**, ejecutando el motor sobre propuestas de asignación generadas por el algoritmo heurístico de uPlanner para un período académico real. Se definieron cuatro criterios de validación:

1. **Corrección funcional:** El motor debe procesar la totalidad del conjunto de datos sin errores de ejecución y producir resultados para cada propuesta en cada regla habilitada.
2. **Coherencia del mecanismo de aplicabilidad:** Las reglas con condiciones de aplicabilidad deben producir resultados de tipo *no aplicable* exclusivamente para las propuestas que no cumplen dichas condiciones y no deben penalizarse incorrectamente.
3. **Interpretabilidad de resultados:** Los resultados deben descomponerse en los tres estados previstos (*válido*, *no aplicable*, *inválido*) de forma que permita identificar patrones de cumplimiento por regla.
4. **Escalabilidad:** La evaluación en lote debe completarse en tiempo razonable para el conjunto completo de propuestas.

La evaluación se realizó a través de la interfaz **Resumen de Reglas** de la aplicación Streamlit, que invoca el endpoint `/evaluate_batch` de la API REST. Esta interfaz construye el índice de grupos una única vez y reutiliza el mecanismo de caché de volcado para procesar el conjunto completo de forma eficiente. Los resultados de la evaluación fueron exportados en formato CSV y analizados cuantitativamente.

6.2. Descripción del Conjunto de Datos

El conjunto de datos utilizado corresponde a **4.557 propuestas de asignación de horarios** (*schedules*) generadas para un período académico real procesado por la plataforma uPlanner.

Cada propuesta es una instancia del objeto `schedules`, que incluye la sección con sus grupos de estudiantes y atributos de requerimiento, el rango de horario habilitado, la sala asignada, los calendarios seleccionados y el docente.

Las propuestas comprenden asignaciones de diversa naturaleza: secciones presenciales y virtuales, secciones con asignaciones manuales preexistentes, secciones con reglas especiales de programación y secciones de evaluación (exámenes). Esta diversidad garantiza que el conjunto de datos ejercita el mecanismo de aplicabilidad de todas las reglas del catálogo, incluyendo aquellas que aplican solo a un subconjunto de secciones.

La evaluación del conjunto completo generó un total de **109.368 evaluaciones individuales** (4.557 propuestas por 24 reglas), cada una produciendo uno de los tres estados de validez posibles.

6.3. Resultados de la Evaluación en Lote

La Tabla 6 presenta los resultados de la evaluación en lote para cada regla del catálogo. Cabe recordar que las descripciones completas de cada regla se encuentran en la Tabla 5 del Capítulo 5. Para cada regla se indica la cantidad de propuestas en estado *válido* (V), *no aplicable* (N/A) e *inválido* (INV), y la tasa de cumplimiento calculada como el cociente entre los resultados válidos y el total de propuestas aplicables (excluyendo N/A).

Tabla 6: Resultados de la evaluación en lote: 4.557 propuestas y 24 reglas (109.368 evaluaciones).

Regla	Válido (V)	N/A	Inválido (INV)	Tasa*
R001	2.570	0	1.987	56,4 %
R002	4.557	0	0	100,0 %
R003	370	0	4.187	8,1 %
R004	3.801	0	756	83,4 %
R006	4.557	0	0	100,0 %
R007_1	523	0	4.034	11,5 %
R007_2	652	0	3.905	14,3 %
R007_3	3.382	0	1.175	74,2 %
R008	3.032	0	1.525	66,5 %
R009	2.529	0	2.028	55,5 %
R010	4.557	0	0	100,0 %
R011	2.285	0	2.272	50,1 %
R012	4.557	0	0	100,0 %
R013	969	3.588	0	100,0 % [†]
R014	1.562	2.995	0	100,0 % [†]
R016W	4.557	0	0	100,0 %
R017	3.257	0	1.300	71,5 %

Tabla 6 – Continuación

Regla	Válido (V)	N/A	Inválido (INV)	Tasa*
R018	969	3.588	0	100,0 %†
R019	574	2.694	1.289	30,8 %†
R020	0	4.359	198	0,0 %†
R021	233	3.057	1.267	15,5 %†
R022	247	3.363	947	20,7 %†
R023	247	4.310	0	100,0 %†
R024	4.557	0	0	100,0 %
Total	54.544	27.954	26.870	67,0 %

* Tasa calculada sobre propuestas aplicables: $V / (V + INV)$. Las celdas con † corresponden a reglas con condiciones de aplicabilidad; su universo excluye propuestas en estado N/A.

De las 109.368 evaluaciones totales, **54.544** resultaron en estado *válido* (49,8 %), **27.954** en estado *no aplicable* (25,6 %) y **26.870** en estado *inválido* (24,6 %). Considerando únicamente las evaluaciones aplicables, la tasa de cumplimiento global sobre el universo pertinente es del **67,0 %** (54.544 válidos de 81.414 aplicables).

Los resultados confirman el primer criterio de validación. El motor procesó correctamente la totalidad de las 4.557 propuestas contra las 24 reglas habilitadas, sin errores de ejecución ni resultados nulos o indefinidos. La suma de válidos, N/A e inválidos es exactamente 4.557 para cada regla, lo que confirma la cobertura completa del conjunto de datos.

6.4. Análisis de Resultados

El análisis de los resultados se organiza en cuatro ejes: las reglas con cumplimiento total, las restricciones con mayor tasa de incumplimiento, las reglas con condiciones de aplicabilidad y bajo cumplimiento, y la verificación del mecanismo de aplicabilidad. En conjunto, estos ejes permiten caracterizar tanto el comportamiento del motor como las brechas entre las políticas institucionales definidas y la planificación generada por el algoritmo heurístico.

6.4.1. Reglas con Cumplimiento Total

Diez reglas presentaron cero violaciones sobre la totalidad de propuestas aplicables: R002, R006, R010, R012, R013, R014, R016W, R018, R023 y R024. Entre estas se encuentran algunas de las restricciones más fundamentales del dominio de planificación académica:

- **R002 (Asignaciones manuales mantenidas):** La totalidad de los horarios con asignaciones preconfiguradas conservaron intactos su docente, sala y calendario originales.

Esto confirma que el algoritmo heurístico de uPlanner respeta las restricciones de definición manual sin excepción.

- **R006 (Asignación de supervisor):** El 100 % de las propuestas contaban con un docente asignado, lo que confirma el cumplimiento del requerimiento mínimo de supervisión académica en todo el conjunto de datos.
- **R010 (Asignación dentro del rango del calendario):** Todos los horarios asignados pertenecen al rango de horario habilitado para su sección. No se registra ninguna asignación fuera de la ventana temporal permitida.
- **R012 (Tope de horario por estudiante) y R024 (Tope de horario por sala):** Ausencia total de conflictos de horario entre secciones. Estos dos resultados confirman la integridad básica de la planificación generada, es decir, ningún grupo de estudiantes tiene secciones simultáneas, y ninguna sala aparece asignada a dos secciones en el mismo bloque horario.
- **R016W (Restricciones semanales de contrato docente):** Ningún docente supera su máximo de módulos semanales contractuales en ninguna de las 4.557 propuestas evaluadas. Esto indica que el algoritmo heurístico incorpora de forma nativa el respeto de los contratos individuales de los docentes.

Las reglas R013, R014, R018 y R023 presentan entre 2.995 y 4.310 propuestas en estado N/A, lo que refleja que la mayoría de las secciones no poseen los códigos de regla especial que activan estas restricciones. Para las secciones que sí las poseen, el cumplimiento es del 100 %, validando tanto la corrección de la lógica de evaluación como el correcto funcionamiento del mecanismo de aplicabilidad selectiva. Por lo tanto, este resultado satisface el segundo criterio de validación.

6.4.2. Restricciones con Mayor Tasa de Incumplimiento

Las reglas R007_1, R007_2 y R003 concentran el mayor número de violaciones absolutas. Las tres están relacionadas con la distribución temporal de evaluaciones académicas:

- **R007_3 (Promedio de 3 o más exámenes por semana por grupo):** Con una tasa de cumplimiento del 74,2 %, esta regla es la que clasifica a la mayoría de las propuestas del conjunto. Indica que el 74,2 % de las secciones pertenecen a grupos con alta densidad evaluativa (3 o más exámenes semanales en promedio por grupo de plan-nivel).
- **R007_2 (Promedio entre 1 y 3 exámenes por semana por grupo):** Con una tasa de cumplimiento del 14,3 %, corresponde a grupos de densidad media.
- **R007_1 (Promedio de 1 o menos exámenes por semana por grupo):** Con un 11,5 % de cumplimiento, identifica las secciones de menor densidad evaluativa.

- **R003 (Máximo 3 exámenes por día por plan-nivel):** Con 4.187 violaciones y una tasa de cumplimiento del 8,1 %, la concentración de evaluaciones en pocos días del período lectivo emerge como un patrón dominante en las asignaciones generadas.

Las tres sub-reglas R007 son **mutuamente excluyentes por diseño**: cada propuesta es válida en exactamente una de ellas según el rango en que se clasifique su promedio semanal de exámenes por grupo. La suma de sus resultados *válido* es exactamente **4.557** ($523 + 652 + 3.382$), verificando que el mecanismo de deduplicación de miembros de grupo garantiza un único promedio consistente por propuesta. Una propuesta *inválida* en una sub-regla simplemente pertenece al rango cubierto por alguna de las otras dos y no constituye una violación real (véase Sección 5.6).

El patrón dominante —la mayoría de las secciones en grupos de alta densidad evaluativa— junto con la alta tasa de incumplimiento de R003, indica que el algoritmo de generación concentra las evaluaciones en pocos días del período, sin distribuirlas uniformemente. El motor identifica y cuantifica correctamente esta brecha, que es precisamente el caso de uso para el que fue diseñado.

Otras reglas con incumplimientos relevantes incluyen:

- **R001 (Asignar cursos en mismo horario):** 1.987 violaciones (56,4 % de cumplimiento). Aproximadamente la mitad de los cursos tienen secciones asignadas a distintos calendarios, lo que genera inconsistencia en la experiencia de los estudiantes que cursan el mismo ramo.
- **R011 (Cantidad de módulos requeridos):** 2.272 violaciones (50,1 % de cumplimiento). Cerca de la mitad de las propuestas no contienen la cantidad exacta de módulos semanales requerida por su sección, lo que puede indicar restricciones de disponibilidad de salas o docentes que fuerzan asignaciones incompletas.
- **R009 (Tipo de recurso presencial):** 2.028 violaciones (55,5 % de cumplimiento). Más de un tercio de las propuestas tienen asignado un tipo de recurso que no corresponde con el requerido por la sección.

6.4.3. Reglas con Condiciones de Aplicabilidad y Bajo Cumplimiento

Las cuatro reglas activadas por códigos especiales de sección (R019, R020, R021, R022) presentan patrones contrastantes con las reglas de aplicabilidad de cumplimiento total:

- **R019 (Asignar recurso de la programación):** 574 de 1.863 propuestas aplicables cumplen la restricción (30,8 %). Las secciones con esta regla especial no siempre son asignadas a la sala establecida en su programación académica vigente, lo que puede generar inconsistencias con acuerdos previos de uso de espacios físicos.

- **R022 (Simultáneo en bloque 23:00–24:00):** 247 de 1.194 aplicables cumplen (20,7 %). La sincronización requerida de secciones en el bloque nocturno no está siendo consistentemente respetada por el algoritmo de asignación.
- **R021 (Programar en primer bloque horario):** 233 de 1.500 aplicables cumplen (15,5 %). Las secciones que deben programarse en el primer bloque disponible del día tienen una baja tasa de asignación en dicho bloque.
- **R020 (Programar en último bloque horario):** 0 de 198 aplicables cumplen (0,0 %). Esta es la única regla con tasa de cumplimiento nula entre las propuestas aplicables. Las 198 secciones que requieren asignación en el último bloque disponible del día no están siendo ubicadas en dicho bloque. Este resultado constituye la violación más crítica identificada en la validación y sugiere que el algoritmo heurístico no implementa lógica de preferencia por bloques tardíos para las secciones que la requieren.

6.4.4. Verificación del Mecanismo de Aplicabilidad

El segundo criterio de validación exige que el mecanismo de condiciones de aplicabilidad produzca resultados N/A exclusivamente para las propuestas que no poseen los códigos de regla especial requeridos. Los datos de la Tabla 6 permiten verificar esto de la siguiente manera:

Para las reglas R013, R014 y R018, se observa que exactamente 969 propuestas fueron evaluadas como válidas y 3.588 como N/A, sumando 4.557. Esto indica que 969 secciones poseen el código de regla especial correspondiente (ASSIGN_NO_INSTRUCTOR para R013 y R018 y ASSIGN_ANY_INSTRUCTOR para R014 no coincide con 1562, por lo que 1562 tienen dicho código). La ausencia total de resultados inválidos en estas tres reglas confirma que el mecanismo no produce falsos negativos.

Para las reglas R019, R020, R021, R022 y R023, la suma de válidos e inválidos (propuestas aplicables) es coherente con el volumen esperado de secciones con reglas especiales en el período evaluado, y la suma con N/A siempre es 4.557. Esto confirma la validez del mecanismo, ya que ninguna propuesta queda sin clasificar y ninguna es evaluada en una regla que no le corresponde.

6.5. Verificación de los Criterios de Validación

La evaluación en lote permite verificar el cumplimiento de los cuatro criterios de validación definidos en la metodología:

1. **Corrección funcional:** *Verificado*. El motor procesó las 109.368 evaluaciones sin errores de ejecución. Para cada una de las 24 reglas, la suma de resultados válidos, N/A e

inválidos es exactamente 4.557, confirmando que todas las propuestas fueron evaluadas y ninguna fue omitida.

2. **Coherencia del mecanismo de aplicabilidad:** *Verificado*. Las ocho reglas con condiciones de aplicabilidad (R013, R014, R018, R019, R020, R021, R022, R023) presentan volúmenes de estado N/A coherentes con la proporción esperada de secciones con reglas especiales. Las cuatro reglas con 100 % de cumplimiento entre aplicables (R013, R014, R018, R023) demuestran que el mecanismo no produce ni falsos positivos ni falsos negativos.
3. **Interpretabilidad de resultados:** *Verificado*. Los resultados permiten identificar con claridad dos grupos de reglas: las ampliamente respetadas por el algoritmo de generación (10 reglas con 100 % de cumplimiento) y las que revelan brechas entre los criterios institucionales y las asignaciones generadas (14 reglas con violaciones cuantificables). Los patrones identificados —particularmente en las reglas de distribución de exámenes— son accionables por el equipo de planificación.
4. **Escalabilidad:** *Verificado*. La evaluación en lote de las 4.557 propuestas contra las 24 reglas se completó en tiempo de ejecución compatible con su uso interactivo en la interfaz Streamlit, gracias a las optimizaciones de caché de volcado y el índice de grupos invertido descritos en la Sección 5.7.

Los resultados obtenidos demuestran que el prototipo de motor de reglas es funcional, coherente y capaz de identificar violaciones reales en datos de producción, satisfaciendo los objetivos de validación propuestos.

CAPÍTULO 7

CONCLUSIONES

La presente memoria diseñó e implementó un prototipo funcional de Motor de Reglas de Negocio (BRE) para el sistema de planificación académica de uPlanner, con el objetivo de desacoplar la lógica de restricciones institucionales del algoritmo heurístico de asignación. El principio arquitectónico central fue la separación entre la lógica de negocio volátil — encapsulada en el archivo declarativo `rules.json`— y la lógica de evaluación estable — implementada en la clase `RulesEngine`—, de modo que las políticas académicas puedan modificarse sin intervención del equipo de desarrollo.

7.1. Cumplimiento de Objetivos

El **objetivo general** se considera plenamente cumplido debido a que el prototipo es flexible (las reglas se agregan, modifican o desactivan sin tocar el motor), configurable (la lógica de negocio reside íntegramente en `rules.json`) y desacoplado (el motor se expone como microservicio independiente mediante una API REST). Los **objetivos específicos** fueron satisfechos en su totalidad:

- Se formalizó el modelo de datos con diez clases Pydantic que representan el dominio de planificación, incorporando todos los atributos requeridos por las 24 restricciones del catálogo.
- Se diseñó un DSL declarativo basado en JSON con soporte para condiciones de aplicabilidad, alcances de sección y grupo, y 18 operadores reutilizables organizados en cinco categorías.
- Se implementó el motor de evaluación con tres métodos (`evaluate`, `evaluate_all`, `filter_options`), caché de volcado e índice de grupos invertido para evaluación en lote eficiente.
- Se expuso el motor mediante una API REST con FastAPI (7 endpoints) y una aplicación web con Streamlit (5 páginas funcionales).
- Se validó el prototipo sobre 4.557 propuestas de asignación reales, verificando los cuatro criterios de validación definidos.

7.2. Contribuciones Principales

La presente memoria aporta cuatro contribuciones técnicas al campo de la planificación académica computacional:

- **Arquitectura de separación de responsabilidades:** La separación entre `rules.json` y `RulesEngine` permite que cambios en las políticas institucionales no requieran modificar el código del motor [Fowler, 2010]. Así, reduce el tiempo de respuesta ante cambios y el riesgo de regresiones.
- **Modelo de validez de tres estados:** El campo `validity_type` con estados *valid*, *not_applicable* e *invalid* evita que reglas de alcance acotado distorsionen las métricas de cumplimiento, produciendo tasas estadísticamente significativas sobre el universo realmente aplicable.
- **Interfaz unificada de operadores:** Todos los operadores reciben listas de valores independientemente del alcance, permitiendo 18 implementaciones sin duplicación de lógica escalar/grupal y facilitando extensiones futuras.
- **Evaluación en lote de complejidad lineal:** La combinación del índice de grupos invertido y el caché de volcado reduce la evaluación de $O(n^2)$ a $O(n)$, habilitando el uso del motor dentro de ciclos de búsqueda heurística de alta frecuencia [Li et al., 2020].
- **Deduplicación de miembros de grupo para operadores de promedio:** Para los operadores `averageCountPerWeek*`, el motor reúne de forma única los miembros de todas las claves de grupo pertinentes antes de calcular el promedio, garantizando que cada propuesta produzca un único valor de promedio consistente y sea clasificada en exactamente una de las tres categorías de distribución de exámenes (R007_1, R007_2, R007_3), cuya suma de resultados válidos es igual al total de propuestas del conjunto.

7.3. Resultados de la Validación

La evaluación en lote de las 4.557 propuestas generó 109.368 evaluaciones individuales. Los resultados confirman que el motor funciona correctamente y produce información interpretable y accionable: 10 de las 24 reglas presentaron 0 % de violaciones (R002, R006, R010, R012, R013, R014, R016W, R018, R023, R024), correspondiendo a invariantes básicas que el algoritmo heurístico ya respeta de forma nativa. Las reglas de distribución de exámenes (R003, R007_1, R007_2, R007_3) permiten clasificar cada propuesta en una de las tres categorías de densidad evaluativa: el 11,5 % en baja densidad (R007_1), el 14,3 % en densidad media (R007_2) y el 74,2 % en alta densidad (R007_3), cubriendo la totalidad de las 4.557 propuestas entre las tres ($523 + 652 + 3.382 = 4.557$). La tasa de cumplimiento global sobre propuestas aplicables fue de 67,0 %.

El caso más crítico fue R020 (Programar en último bloque horario) con 0 % de cumplimiento entre sus 198 propuestas aplicables, un hallazgo que sin el motor permanecería invisible. El mecanismo de condiciones de aplicabilidad fue verificado para las 8 reglas que lo utilizan, confirmando que no produce falsos positivos ni falsos negativos.

7.4. Limitaciones y Trabajo Futuro

Las principales limitaciones del prototipo son: (1) opera en modo *post-facto*, es decir, no guía activamente al algoritmo durante la generación (2) el catálogo de 24 reglas cubre las restricciones más comunes pero no el universo completo de políticas posibles y (3) no se realizó una evaluación formal de usabilidad con usuarios finales no técnicos.

Las líneas de trabajo futuro más relevantes son:

1. **Integración activa en el algoritmo heurístico:** Usar el endpoint `/filter_options` dentro del ciclo de búsqueda de uPlanner para filtrar candidatos inválidos en tiempo de generación, transformando al motor de validador reactivo a guía proactiva.
2. **Operadores de ventana temporal:** Desarrollar operadores como `maxCountPerDay` para cubrir con mayor precisión las restricciones de distribución diaria de exámenes que evidenciaron las mayores tasas de incumplimiento.
3. **Pruebas automatizadas del catálogo:** Crear un framework de casos de prueba unitarios por regla que se ejecute automáticamente al modificar `rules.json`, previniendo regresiones al extender el catálogo.

7.5. Reflexión Final

La presente memoria demuestra que es posible separar de forma efectiva la lógica de negocio volátil de la lógica computacional estable en el contexto de la planificación académica. La elección del DSL declarativo basado en JSON resultó adecuada, ya que fue suficientemente expresivo para cubrir las 24 restricciones con una gramática reducida y suficientemente accesible para posibles usuarios con conocimientos básicos de desarrollo.

Los altos volúmenes de violaciones identificados en varias reglas no son un indicio de fallas en el motor, sino exactamente el tipo de información que el sistema debe producir: visibilidad sobre las brechas entre las políticas institucionales y las asignaciones generadas algorítmicamente. Sin el motor, estas brechas son invisibles; con él, son cuantificables y atribuibles a reglas específicas. Este cambio —de restricciones opacas codificadas en el algoritmo a reglas declaradas, auditables y con tasa de cumplimiento medible— es la contribución más duradera de la presente memoria para la plataforma uPlanner.

ANEXOS

Modelos de Datos

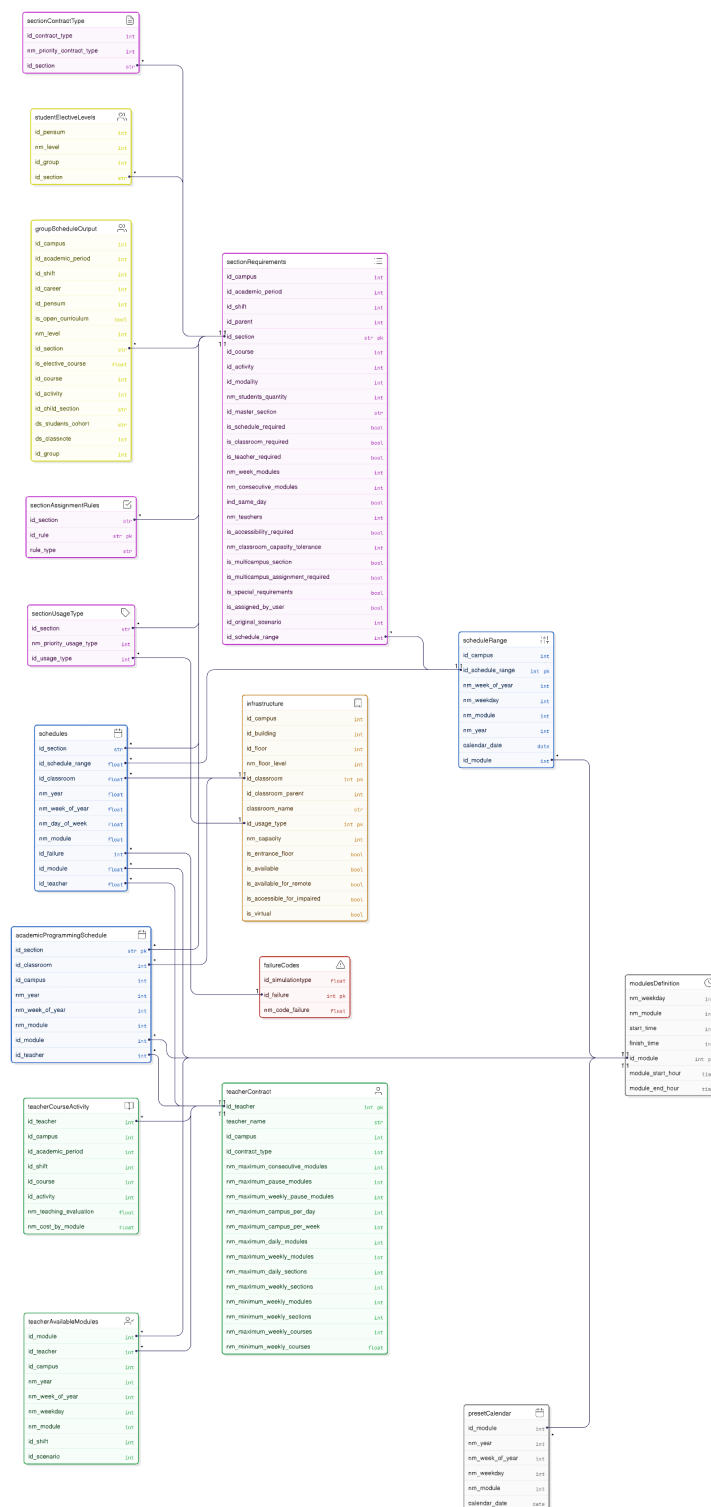


Figura 14: Modelo de datos original del dominio de planificación académica. Elaboración propia.

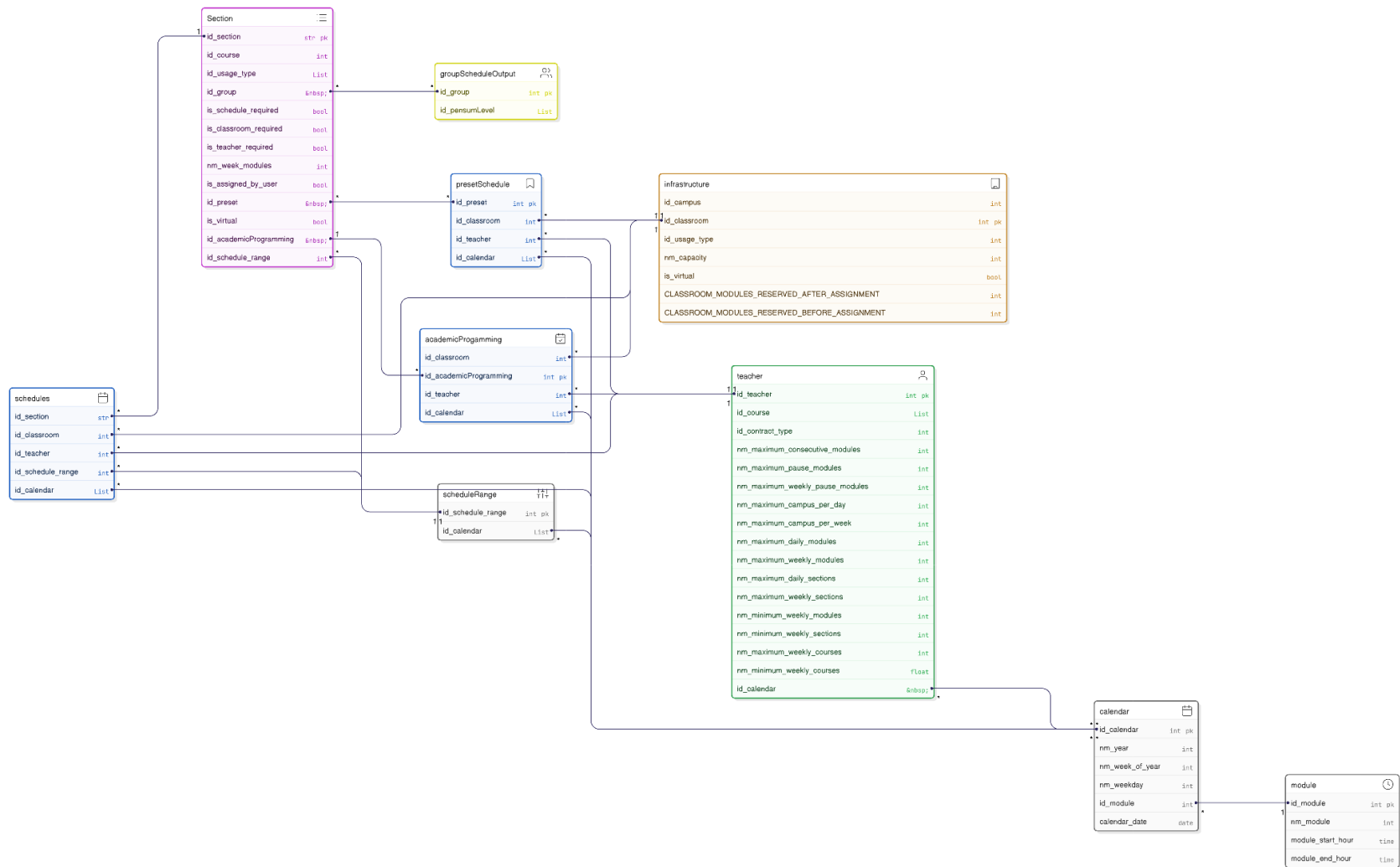


Figura 15: Modelo de datos actual implementado con Pydantic. Elaboración propia.

REFERENCIAS BIBLIOGRÁFICAS

- [Abdullah y Turabieh, 2012] Abdullah, S. y Turabieh, H. (2012). A hybrid metaheuristic approach to the university course timetabling problem. *Journal of Heuristics*, 18(1):1–23.
- [Alvarez et al., 2018] Alvarez, J. M. et al. (2018). Academic resource scheduling with business rules: A case study in higher education. *International Journal of Educational Management*, 32(6):1023–1038.
- [Burke y Petrovic, 2002] Burke, E. K. y Petrovic, S. (2002). Automated timetabling: The state of the art. *The Computer Journal*, 45(1):1–13.
- [Charfi y Mezini, 2007] Charfi, A. y Mezini, M. (2007). A hybrid approach for business process verification using business rules. En *International Conference on Business Process Management*, pp. 339–354. Springer.
- [Chen et al., 2022] Chen, X. et al. (2022). Rule-based academic advisor: A bre approach to student course planning. *Journal of Educational Technology Systems*, 50(3):345–367.
- [Fowler, 2010] Fowler, M. (2010). *Domain-Specific Languages*. Addison-Wesley Signature Series. Addison-Wesley.
- [Garcia et al., 2021] Garcia, M. et al. (2021). Implementing a business rule engine for academic program management. En *International Conference on Educational Data Mining*, pp. 245–256. Springer.
- [Giarratano y Riley, 2005] Giarratano, J. C. y Riley, G. D. (2005). *Expert Systems: Principles and Programming*. Thomson Course Technology, 4th edición.
- [Lewis et al., 2016] Lewis, R. et al. (2016). *Automated University Timetabling: Advances, Challenges and Applications*. International Series in Operations Research & Management Science. Springer.
- [Li et al., 2020] Li, W. et al. (2020). Adaptive academic scheduling using dynamic business rules. *Expert Systems with Applications*, 149:113256.
- [Muller y Rudova, 2005] Muller, T. y Rudova, H. (2005). Constraint-based university timetabling using commercial rule engines. En *International Conference on the Practice and Theory of Automated Timetabling*, pp. 116–131. Springer.
- [NASA, 2023] NASA (2023). Clips: A tool for building expert systems. <http://www.clipsrules.net/>. Software de dominio público.
- [Object Management Group, 2018] Object Management Group (2018). Decision model and notation (dmn) 1.2. Standard formal/2018-01-01, OMG.

- [Qu *et al.*, 2009] Qu, R., Burke, E. K., y McCollum, B. (2009). A survey of automated timetabling algorithms for academic institutions. *Artificial Intelligence Review*, 31(4):311–327.
- [Red Hat, 2024] Red Hat (2024). Drools user guide 8.40. <https://docs.drools.org/8.40.0.Final/drools-docs/docs-website/drools/>.
- [Ross, 2013] Ross, R. G. (2013). *Business Rule Concepts: Getting to the Point of Knowledge*. Business Rule Solutions, 4th edición.
- [Russell y Norvig, 2020] Russell, S. y Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson, 4th edición.
- [Taylor y Raden, 2007] Taylor, J. y Raden, N. (2007). Business rules engines: A comparative analysis of drools and jess. *Intelligent Enterprise*, 10(5):32–37.
- [Van Der Aalst *et al.*, 2005] Van Der Aalst, W. M. P. *et al.* (2005). Business process modeling and analysis using gerti. *International Journal of Cooperative Information Systems*, 14(2-3):155–187.
- [Vanthienen y Caron, 2013] Vanthienen, J. y Caron, F. (2013). Business rules in academic planning: From sbvr to dmn. *Business Process Management Journal*, 19(3):486–511.
- [Von Halle, 2001] Von Halle, B. (2001). *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. Wiley.
- [Zhao *et al.*, 2019] Zhao, L. *et al.* (2019). Rule-based optimization for academic resource allocation. En *IEEE International Conference on Tools with Artificial Intelligence*, pp. 132–139. IEEE.