Universitat Politècnica de València

Cátedra ENIA-UPV en IA Desarrollo Sostenible - NUNSYS

Sistema Multi-Agente Jerárquico para optimizar el IMC del Proyecto NARRATE

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Autor:
Álvaro Prado Expósito

Índice

1. Presentación de la propuesta

1.1. Objetivo Principal

El presente documento constituye la memoria de las prácticas extracurriculares realizadas en la empresa NUNSYS, enmarcadas dentro de la continuidad del proyecto europeo NARRATE. El objetivo principal de este periodo de prácticas ha sido evolucionar y consolidar el diseño, implementación y validación de una arquitectura multi-agente jerárquica y descentralizada para el orquestador cognitivo *Intelligent Manufacturing Custodian* (IMC).

El propósito de este trabajo ha sido dar solución a las deficiencias detectadas en versiones previas del sistema, las cuales operaban bajo una arquitectura monolítica (basada en cadenas lineales) que sufría de saturación cognitiva, rigidez procedimental y alucinaciones técnicas. Para superar estas barreras, el trabajo se ha centrado en desarrollar un sistema de sub-agentes inteligentes especializados en el consumo de APIs, orquestados centralmente por un agente supervisor fundamentado en el paradigma ReAct y en el uso de grafos de estado.

1.2. Arquitectura de Tool Agents

Tras evaluar diversas arquitecturas, se determinó que la estrategia más efectiva para un entorno industrial y logístico de la complejidad de NARRATE consiste en definir a los sub-agentes como herramientas (*tools*) del agente supervisor bajo el principio de *agent-as-a-tool*.

Bajo este enfoque de aislamiento perimetral de contextos, el Agente Supervisor asume en exclusiva la planificación estratégica y se abstrae de la complejidad procedimental de los protocolos web. Cuando este supervisor requiere interactuar con una API específica (por ejemplo, para extraer la estructura de un producto o evaluar un riesgo logístico), delega la tarea de red en el "Tool Agent" especializado correspondiente. Esta abstracción permite al agente principal tratar a los sub-agentes como funciones ejecutables y deterministas, mitigando por completo el desbordamiento de la ventana de contexto (Context Window Saturation) que colapsaba al sistema anterior.

La arquitectura se complementa con una fuerte colaboración neuro-simbólica. El motor neuronal probabilístico (LLM) interactúa con un ecosistema simbólico determinista (parsers, validadores de esquemas, bases de datos) que aporta exactitud técnica y evita desviaciones en las ejecuciones de los microservicios.

1.3. Modo de Uso

La interacción con el sistema se orquesta mediante un flujo de trabajo estructurado en el que la entrada de información es interceptada por un Enrutador de Consultas (Query Router) que aplica un modelo de clasificación semántica *Zero-shot*. El sistema procesa diversas tipologías de entrada:

- **Alerta IoT:** Señales técnicas automáticas procedentes de los sensores de la fábrica. El enrutador desvía el flujo hacia un módulo especializado (RAG_MOCK) que sintetiza explicaciones basadas en manuales operativos.
- **Consulta Libre (Orquestación Dinámica):** El mensaje plantea un requerimiento complejo sobre la cadena de suministro. El Agente Supervisor asume el control, diseña un plan iterativo y asigna las subtareas a sus sub-agentes especializados.
- **Protocolo de Emergencia (System Override):** Si el enrutador detecta que la entrada coincide con un Tipo de Riesgo predefinido, se anula la autonomía exploratoria del supervisor. Se inyecta un plan de respuesta estricto forzando una ejecución secuencial e ineludible.

2. Vías de desarrollo

Durante la continuación del desarrollo del código en este periodo extracurricular, han ido apareciendo diferentes retos operativos y arquitectónicos que exigían un rediseño profundo del ecosistema agéntico y analítico. A continuación, se detallan los avances realizados.

2.1. Generación de herramientas dinámicas

Para dotar al sistema de la capacidad de interactuar con un entorno de microservicios en constante cambio, se consolidó un mecanismo de especialización dinámica basado en el patrón *Factory*. Este desarrollo abstrae la red y opera de forma independiente:

2.1.1. Descubrimiento y Factoría Semántica

El Agente Supervisor cuenta con un registro centralizado que traduce especificaciones OpenAPI o Swagger en nodos de ejecución. El proceso de destilación semántica incluye:

- **Extracción de entidades y filtrado:** Identificación de *endpoints* clave, excluyendo el ruido y mapeando esquemas de datos simplificados para reducir el consumo de tokens y evitar alucinaciones paramétricas.
- **Generación de *Cheat Sheets*:** Compilación de una guía rápida inyectada directamente en el contexto del sub-agente (*System Prompt*), limitando los métodos HTTP válidos y las estructuras de *payload*.

2.1.2. Pipeline de Peticiones y Validación Simbólica

Una vez instanciado, el sub-agente autónomo (ejecutado en su propio sub-grafo) procesa la instrucción en lenguaje natural. Antes de lanzar la petición web, el flujo atraviesa una validación estricta utilizando la librería Pydantic. Esta barrera simbólica asegura que los tipos de datos coinciden exactamente con el contrato de la API. Si la API devuelve un código de error, el sub-agente intercepta el fallo y ejecuta un mecanismo de *Smart Fallback* o autocorrección antes de reportar un fallo irrecuperable al supervisor.

2.2. Mitigación de Alucinaciones y Optimización de la Planificación

Uno de los problemas más severos heredados de la versión anterior era el comportamiento estocástico del modelo: inventaba parámetros, alucinaba rutas de API y entraba en bucles infinitos. Durante estas prácticas, se abordó este problema implementando técnicas avanzadas de *Prompt Engineering* y políticas de ejecución restrictivas.

2.2.1. Políticas de Schema Awareness y Zero Guessing

Se introdujo una regla inquebrantable en la orquestación: la prohibición absoluta de adivinar o autocompletar identificadores. Si el supervisor recibe un nombre descriptivo de un proveedor (ej. “Bespoke Baby Cot”), está obligado a realizar un *Pre-flight Resolution*; es decir, debe primero invocar a un sub-agente de inventario estructural (Neo4j) para obtener el ID exacto antes de delegar cualquier tarea analítica.

2.2.2. Modificaciones Clave en el Prompt

Se reescribió el *Prompt Template* para forzar una cadena de razonamiento (Chain of Thought) verificable. Se incluyó una validación previa crítica:

1. Identificar parámetros obligatorios requeridos por la tarea.
2. Verificar explícitamente si se posee dicho dato (SÍ/NO).
3. Si falta información, está prohibido inventar IDs y se debe paralizar el avance para buscar la dependencia faltante.

Adicionalmente, se consolidó un *Batching Protocol* (procesamiento por lotes). Si el agente necesita evaluar a múltiples proveedores, tiene prohibido iterar secuencialmente en bucle; debe consolidar todos los IDs en una única petición agrupada, reduciendo drásticamente la latencia y el consumo computacional.

2.3. Implementación de Kong Gateway: Seguridad y Trazabilidad

Para evolucionar hacia una arquitectura centralizada y estandarizada, se integró y configuró Kong Gateway como proxy inverso y punto único de acceso (API Gateway) para todos los microservicios.

2.3.1. Despliegue y Control de Acceso

El despliegue se realizó mediante Docker en modo *DB-less*, favoreciendo configuraciones declarativas alineadas con prácticas GitOps. Se estructuró la seguridad mediante políticas RBAC, utilizando *API Keys* ocultas para el agente (inyectadas mediante clausuras estáticas para evitar filtraciones por *prompt injection*) y limitación de tasa (rate-limiting a 100 req/min) para prevenir ataques DoS o bucles accidentales de los agentes.

2.3.2. Observabilidad (Jaeger y Zipkin)

Para garantizar la explicabilidad requerida en el proyecto, se configuró una telemetría distribuida. Se activaron *plugins* de identificación (`correlation-id`) y se conectó el Gateway a Jaeger mediante Zipkin con un ratio de muestreo del 100 %. Cada acción de los sub-agentes incluye una firma de telemetría inyectada en su respuesta (metadatos de *endpoints* ejecutados), otorgando al Agente Supervisor y a la plataforma un registro de auditoría transparente.

2.4. Transición de LangChain a LangGraph: Evolución hacia una Arquitectura Determinista

El avance arquitectónico más significativo del periodo ha sido la migración desde las cadenas secuenciales monolíticas (*AgentExecutor* de LangChain) hacia grafos de estado cíclicos implementados con **LangGraph**. Esta decisión tecnológica radicó en tres pilares:

- **Native Tool Calling:** Se abandonó el arcaico parseo de texto mediante expresiones regulares en favor de métodos estructurados (`bind_tools` y `ToolNode`). Al vincular esquemas Pydantic directamente a la API del modelo, se eliminó la "alucinación sintáctica".
- **Control de Estado (StateGraph):** LangGraph permite concebir al agente no como un simple generador de texto, sino como una máquina de estados finitos. La memoria se gestiona de forma aislada por cada hilo de ejecución (`MemorySaver`), lo que permite transiciones condicionales. Si una llamada falla, la arista condicional devuelve el flujo al nodo de razonamiento para su inmediata autocorrección sin que el sistema colapse.
- **Eficiencia:** La nueva arquitectura redujo los tiempos de latencia y eliminó los reprocesamientos innecesarios, estabilizando el consumo de *tokens* a pesar de la alta complejidad logística.

2.5. Evaluación del Rendimiento del Sistema

Las pruebas empíricas ejecutadas durante las prácticas demostraron la superioridad aplastante de la nueva arquitectura frente al modelo monolítico previo. A lo largo de escenarios complejos de simulación condicionada y exploración dinámica, la versión anterior obtenía tasas de completitud (TCR) del 0 %, sumida en bucles y fallos de formato. La arquitectura actual implementada con LangGraph alcanzó un **Task Completion Rate (TCR) del 100 %** en todos los escenarios, reduciendo drásticamente la varianza y estabilizando la latencia total del proceso en promedios óptimos (ej. 12-15 segundos). La dispersión del consumo de tokens se agrupó en un clúster compacto y predecible, erradicando el gasto computacional descontrolado.

3. Aprendizaje

El trabajo realizado durante estas prácticas extracurriculares ha supuesto una profunda inmersión en la ingeniería de Inteligencia Artificial moderna y el diseño de arquitecturas de software distribuidas. Los aprendizajes más destacados son:

- **Evolución de las Arquitecturas Agénticas:** Se constató empíricamente la ineficacia de los sistemas monolíticos (*Plan-and-Execute*) frente a entornos con múltiples APIs. La adopción de grafos de estado (*LangGraph*) demostró ser el estándar operativo actual necesario para crear sistemas robustos, cíclicos y con capacidades de autorreparación.
- **Sinergia Neuro-Simbólica:** Se asimiló la criticidad de combinar el razonamiento probabilístico de un LLM con capas simbólicas deterministas (Pydantic, validadores OpenAPI). Otorgar libertad absoluta a un modelo genera alucinaciones; confinarlo con esquemas de datos estrictos y validación de tipos asegura la viabilidad del producto en la Industria 4.0.
- **Orquestación Cognitiva y Prompting Avanzado:** El desarrollo evidenció que las técnicas de *Few-Shot Prompting* e imposición de reglas estructurales (como el *Pre-flight Resolution* y el *Zero Guessing*) son imprescindibles para gobernar el comportamiento de los agentes.
- **Infraestructura y Seguridad:** La configuración de Kong API Gateway y la telemetría distribuida (Jaeger) aportaron una visión madura sobre cómo securizar, exponer y monitorizar aplicaciones basadas en microservicios a nivel empresarial.

4. Futuras líneas de mejora

Con base en los logros consolidados, se identifican las siguientes líneas de trabajo futuro acordes a las exigencias continuas del Proyecto NARRATE:

- **Optimización del Consumo de Tokens:** Implementar técnicas de *Prompt Caching* y evaluar la integración de Modelos de Lenguaje de menor escala (Small Language Models o SLMs), ajustados localmente (fine-tuning), para delegarles tareas cognitivamente menos exigentes dentro de los sub-agentes.
- **Adaptación al Ecosistema de Datos:** Actualizar los módulos de inyección dinámica para soportar las nuevas versiones de los *Blueprints* y los *Digital Twins* que se irán definiendo dentro del consorcio NARRATE.
- **Integración IoT en Tiempo Real:** Trascender los entornos simulados conectando la capacidad de orquestación del IMC directamente a flujos masivos de telemetría física real (MQTT), evaluando su comportamiento ante picos de emergencias concurrentes en la planta.
- **Desarrollo de GraphRAG:** Evolucionar el sistema de apoyo RAG actual (*RAG_MOCK*) hacia tuberías de *Retrieval-Augmented Generation* basadas en grafos semánticos, capaces de recuperar manuales técnicos complejos respetando las dependencias causales de las redes de fabricación.