Facsímil 05

Agentes y orquestación

Capítulo 12

Lo que deberías saber: agentes y orquestación

686f6c61 · 2026

Capítulo 12: Lo que deberías saber: agentes y orquestación

Cerrar el facsímil sin cerrar la pregunta

Este facsímil empezó con una duda práctica: ¿cuándo merece la pena llamar agente a un sistema y cuándo basta con un prompt, una función o una interfaz más clara?

La respuesta no era una etiqueta. Era una arquitectura. Un agente útil no es “un modelo con herramientas”. Es un sistema que mantiene estado, decide acciones, observa resultados, respeta permisos, registra trazas, puede pedir revisión, se integra con otros sistemas y se evalúa con casos repetibles.

Si has entendido el facsímil, deberías poder mirar una demo de agente y preguntar: qué estado conserva, qué tools puede llamar, qué efecto tienen, qué permisos gobiernan esas acciones, cómo se recupera si algo falla, qué traza deja, qué coste tiene y qué gate decide si una versión avanza.

Fecha de corte y alcance

Fecha de corte: 10 de junio de 2026.
Alcance: este cierre resume conceptos estables del facsímil: agente como bucle estado-acción-observación, tools como contratos, memoria como sistema separado del prompt, harness, permisos, SDKs, MCP, A2A, routing y evaluación de trayectorias.

Los nombres de SDKs, modelos, APIs, precios y protocolos se moverán. Lo que queremos conservar es más estable: un agente es software con decisiones observables. Por tanto, se diseña, se versiona, se prueba, se mide y se opera.

La frase que resume el facsímil

Un agente operable es la suma de varias piezas. Es un mnemónico para no olvidar ninguna, no una ecuación de la literatura: modelo, estado, tools, permisos, handoffs, observabilidad y evaluación.

SÍMBOLO	SIGNIFICADO	EJEMPLO
A	Agente operable.	Asistente que revisa una cita, consulta fuente, valida APA y pide aprobación si publica.
M	Modelo.	LLM que interpreta instrucciones y genera decisiones.
S	Estado.	Conversación, memoria, plan, tareas pendientes, contexto compacto.
T	Tools.	Buscar fuente, validar formato, crear ticket, leer base de datos.
P	Permisos.	Qué puede leer, escribir, ejecutar o pedir a una persona.
H	Harness.	Entorno que limita, ejecuta, observa y captura trazas.
O	Orquestación.	Routing, handoffs, MCP, A2A, workflows y subagentes.
E	Evaluación.	Dataset, trayectorias, coste, latencia, gates y regresiones.

La fórmula no pretende ser matemática profunda. Sirve como recordatorio: si falta una pieza, la palabra “agente” puede estar escondiendo una demo frágil.

Lo que ya no deberías confundir

CONFUSIÓN	FORMA PRECISA DE DECIRLO
“Un agente es un LLM que responde”.	Un agente observa, decide, actúa, registra y vuelve a decidir.
“Memoria es meter más contexto”.	Memoria implica qué se guarda, cuándo, dónde, con qué permisos y cómo se recupera.
“Una tool es una función cualquiera”.	Una tool tiene schema, semántica, efecto, permisos, errores y observabilidad.
“El SDK es la arquitectura”.	El SDK implementa patrones; la arquitectura la decides tú.
“Si pasa la demo, funciona”.	Funciona si pasa casos, trazas, coste, permisos, latencia y regresiones.
“Orquestar es llamar muchos agentes”.	Orquestar es decidir ruta, contrato, handoff, responsabilidad y evaluación.

Wooldridge y Jennings definían los agentes como sistemas situados en un entorno, capaces de actuar de forma autónoma para cumplir objetivos.[1] Ese vocabulario clásico sigue siendo útil, pero ahora lo aterrizamos en modelos, tools, trazas y sistemas distribuidos.

Lo que faltaría si lo revisa un ingeniero informático

Para una persona de ingeniería informática, el facsímil no debería cerrar con “ya sé qué es un agente”. Debería cerrar con una lista de condiciones para llevarlo a un sistema mantenible. Un agente productivo se parece menos a una conversación y más a un servicio distribuido que llama otros servicios, mantiene estado, falla parcialmente, consume presupuesto y deja evidencia.

TEMA QUE AÑADIRÍA AL CIERRE	PREGUNTA QUE DEBE RESPONDER	POR QUÉ IMPORTA
Máquina de estados	¿En qué estados puede estar una run y qué transiciones son válidas?	Evita flujos implícitos imposibles de depurar.
Idempotencia	¿Qué pasa si llega dos veces la misma petición?	Evita duplicar tickets, publicaciones, cobros o acciones persistentes.
Semántica de reintentos	¿Qué errores se reintentan, cuántas veces y con qué espera?	Un retry mal diseñado multiplica coste y efectos.
Timeouts y cancelación	¿Cuándo se aborta una tool o una run completa?	Sin límites, el sistema se atasca y consume recursos.
Colas y backpressure	¿Qué ocurre si llegan más tareas de las que podemos atender?	Protege latencia y evita saturar tools externas.
Versionado de contratos	¿Qué versión de prompt, tool, schema, policy y dataset produjo esta salida?	Sin versión no hay rollback ni comparación honesta.
Consistencia del estado	¿Qué se guarda antes y después de cada acción?	Evita que una traza diga una cosa y la base de datos otra.
Observabilidad estándar	¿Hay `trace_id`, `span_id`, atributos, eventos y propagación de contexto?	Permite seguir una run aunque atraviese varios servicios.
SLO y presupuesto	¿Cuál es el p95 aceptable, coste máximo y tasa de fallo tolerable?	Sin objetivos no hay operación, solo impresiones.
CI/CD de agentes	¿Qué evals bloquean un PR, una nightly o una publicación?	Convierte “parece mejor” en decisión revisable.

OpenTelemetry define APIs para crear spans y trazas, y W3C Trace Context estandariza cómo propagar contexto entre servicios.[2][3] En un agente, esto significa que una decisión del router, una llamada al modelo, una tool MCP, una cola de revisión y un gate de evaluación pueden formar parte de la misma historia técnica.

El versionado semántico ayuda a distinguir cambios compatibles de cambios que rompen contrato.[4] En agentes, no solo versionamos librerías: versionamos prompts, tools, policies, datasets, modelos y formatos de traza.

La ingeniería de ML ya avisaba de una deuda técnica específica: sistemas con modelos pueden esconder dependencias, configuraciones, datos y comportamiento de difícil mantenimiento.[5] Amershi y colaboradores muestran que desarrollar sistemas de ML exige prácticas de ingeniería distintas a las de software clásico, especialmente por datos, experimentación y evaluación continua.[6] TFX es un buen ejemplo histórico de plataforma pensada para producción: no basta entrenar o ejecutar; hay que validar datos, modelos y despliegues.[7]

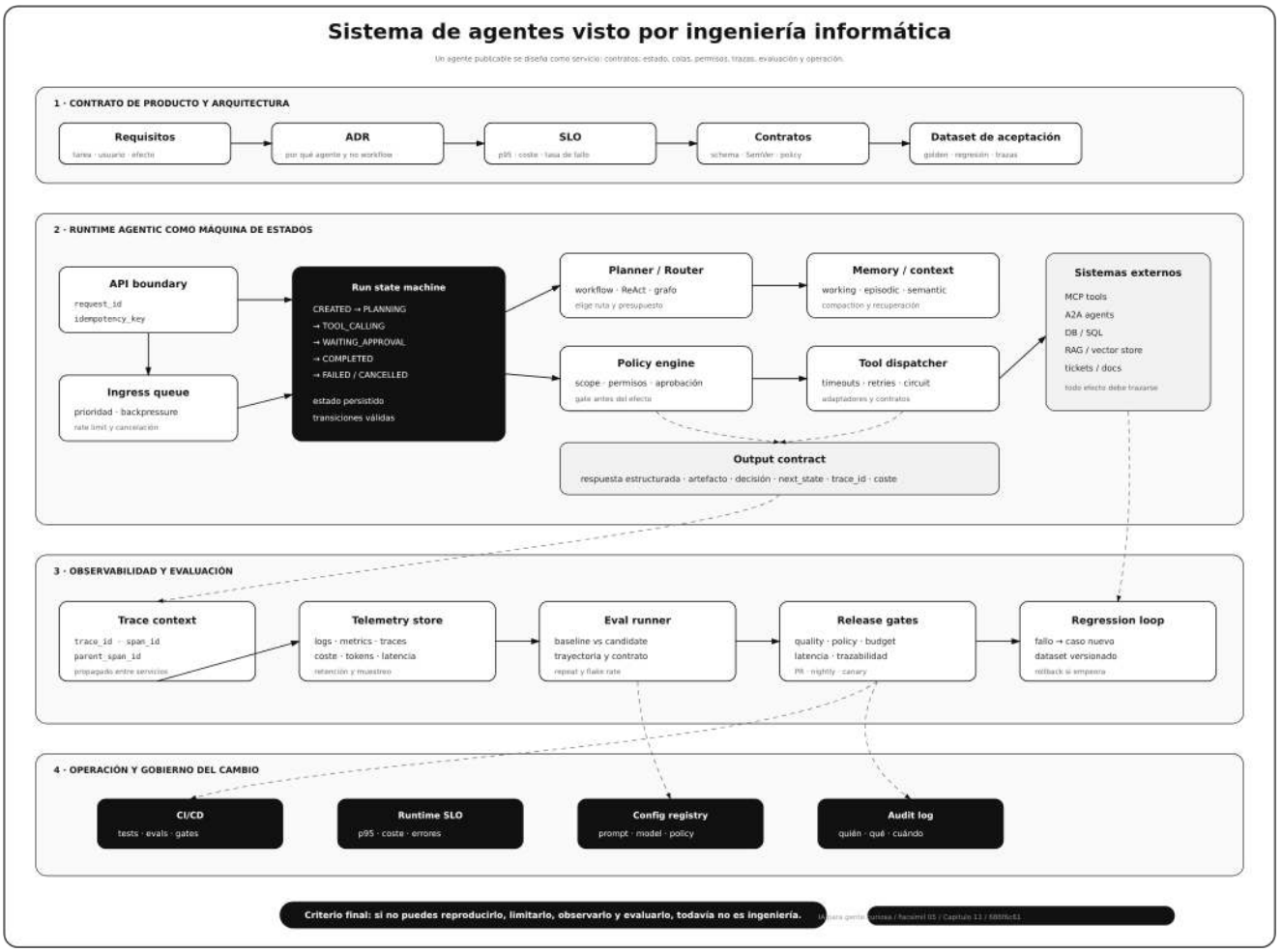
Recapitulación activa por capítulos

Esta tabla no es un índice. Es una prueba rápida de criterio. Si no puedes explicar la columna derecha, vuelve al capítulo correspondiente.

CAPÍTULO	QUÉ DEBERÍAS PODER DEFENDER	PREGUNTA DE CONTROL
01	Elegir entre prompt, workflow y agente.	¿Hay bucle, estado, tools y objetivo o solo una respuesta?
02	Describir estado, acción, observación y política.	¿Qué cambia después de cada paso?
03	Diseñar tools con contratos operativos.	¿Qué argumentos, errores, efectos y permisos tiene la tool?
04	Separar contexto, memoria, compaction y handoff.	¿Qué se guarda y qué se vuelve a pasar al modelo?
05	Elegir arquitectura agentic según tarea.	¿ReAct, planificador, workflow, multiagente o grafo?
06	Construir harness con límites, sensores y trazas.	¿Cómo se observa y reproduce una ejecución?
07	Integrar SDKs sin delegar el diseño.	¿Qué es portable y qué es específico del proveedor?
08	Diseñar permisos y revisión humana.	¿Qué acciones requieren aprobación y por qué?
09	Orquestar routing, MCP, A2A y workflows.	¿Quién decide, quién actúa y qué contrato viaja?
10	Evaluar trayectoria, coste y gates.	¿Cómo sabes que la versión nueva mejora de verdad?
11	Montar el bucle con verificador: propón, mide y refina.	¿La observación es una señal dura (legal y medible) o solo texto plausible?

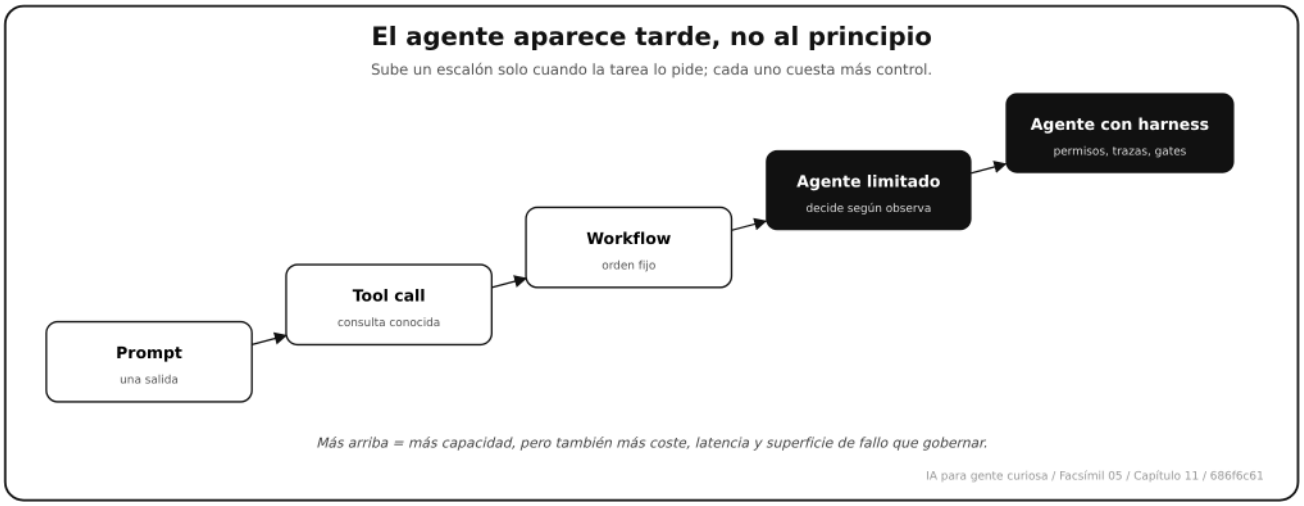
ReAct popularizó el patrón de intercalar razonamiento y acciones observables con tools, en vez de producir una respuesta de una sola vez.[8] ToolLLM mostró la importancia de enseñar y evaluar uso de APIs reales en modelos de lenguaje.[9]

Mapa visual del facsímil



La decisión técnica: de tarea a arquitectura

Todo el facsímil cabe en una idea: un agente es un POMDP (un proceso de decisión bajo observación parcial) que un modelo aproxima, y cada capítulo fue haciendo ese POMDP operable. El estado se volvió contexto y memoria; las acciones, tools con contrato; las observaciones, salidas tipadas; la recompensa, un gate de evaluación; y los límites, presupuesto y permisos. La pregunta de diseño no es «¿uso un agente?», sino «¿qué escalón necesita esta tarea?».



Cuando alguien pide “un agente”, la respuesta profesional no es sí o no. Es ordenar la decisión.

PREGUNTA	SI LA RESPUESTA ES SÍ	SI LA RESPUESTA ES NO
¿La tarea requiere varios pasos?	Puede tener sentido un workflow o agente.	Empieza por prompt, función o interfaz.
¿Hay tools con efectos reales?	Diseña contrato, permiso y traza.	No vendas autonomía que no existe.
¿El sistema necesita recordar algo?	Separa memoria, contexto y almacenamiento.	No metas historial infinito en prompt.
¿Hay varias rutas posibles?	Añade routing explícito y métrica de ruta.	Mantén camino simple y evaluable.
¿Puede afectar a datos o personas?	Añade approval y gate de runtime.	Aun así registra la decisión.
¿Puedes evaluar la trayectoria?	Versiona dataset y traces.	Todavía estás en fase exploratoria.

Smith ya describió el Contract Net Protocol como coordinación distribuida entre participantes que anuncian tareas, reciben propuestas y asignan trabajo.[10] La idea resuena con sistemas modernos: incluso cuando usamos LLMs, coordinar trabajo exige contratos y responsabilidad.

Jennings, Sycara y Wooldridge insistían en que la investigación de agentes no iba solo de piezas aisladas, sino de coordinación, interacción, entornos y metodologías de desarrollo.[11] Esa advertencia encaja perfectamente con este facsímil: si un agente moderno no deja claro cómo coordina, cómo observa y cómo se evalúa, todavía no está bien diseñado.

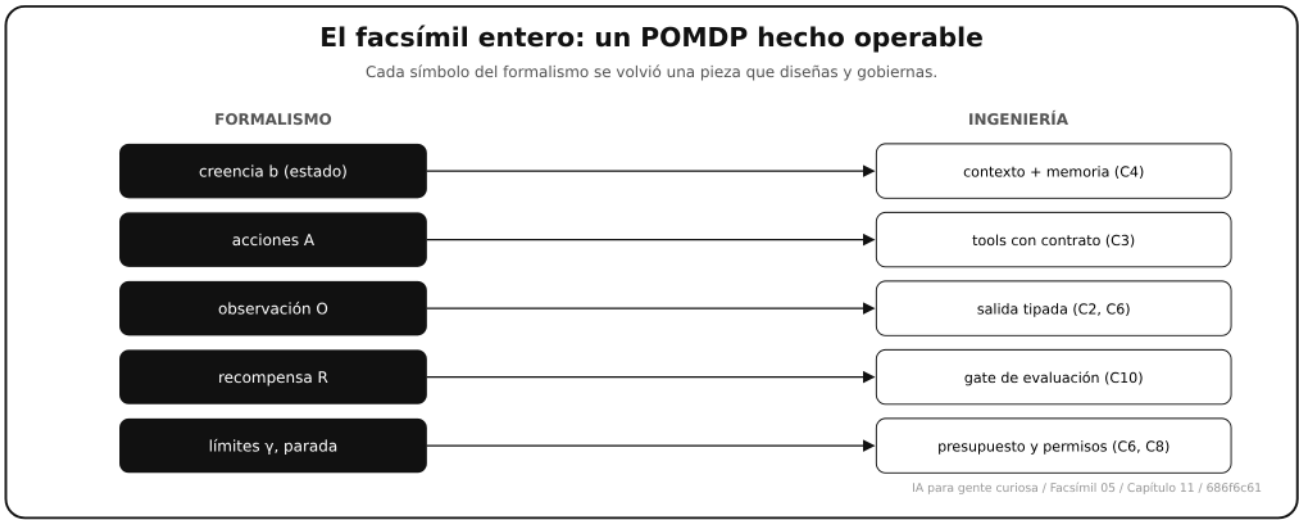
El cambio de mentalidad que cierra el facsímil

Si hubiera que resumir todo este facsímil en una sola frase, no sería técnica, sería un cambio de mentalidad. Se empieza creyendo que construir un agente consiste en encontrar el modelo lo bastante bueno y el prompt lo bastante astuto para que «lo resuelva solo». Se termina entendiendo lo contrario: que la capacidad del modelo es el punto de partida, no el destino, y que el trabajo de ingeniería consiste en diseñar el sistema que rodea al modelo para que su capacidad se vuelva fiable, gobernable y repetible. Esa inversión es la que separa a quien hace demos de quien construye sistemas.

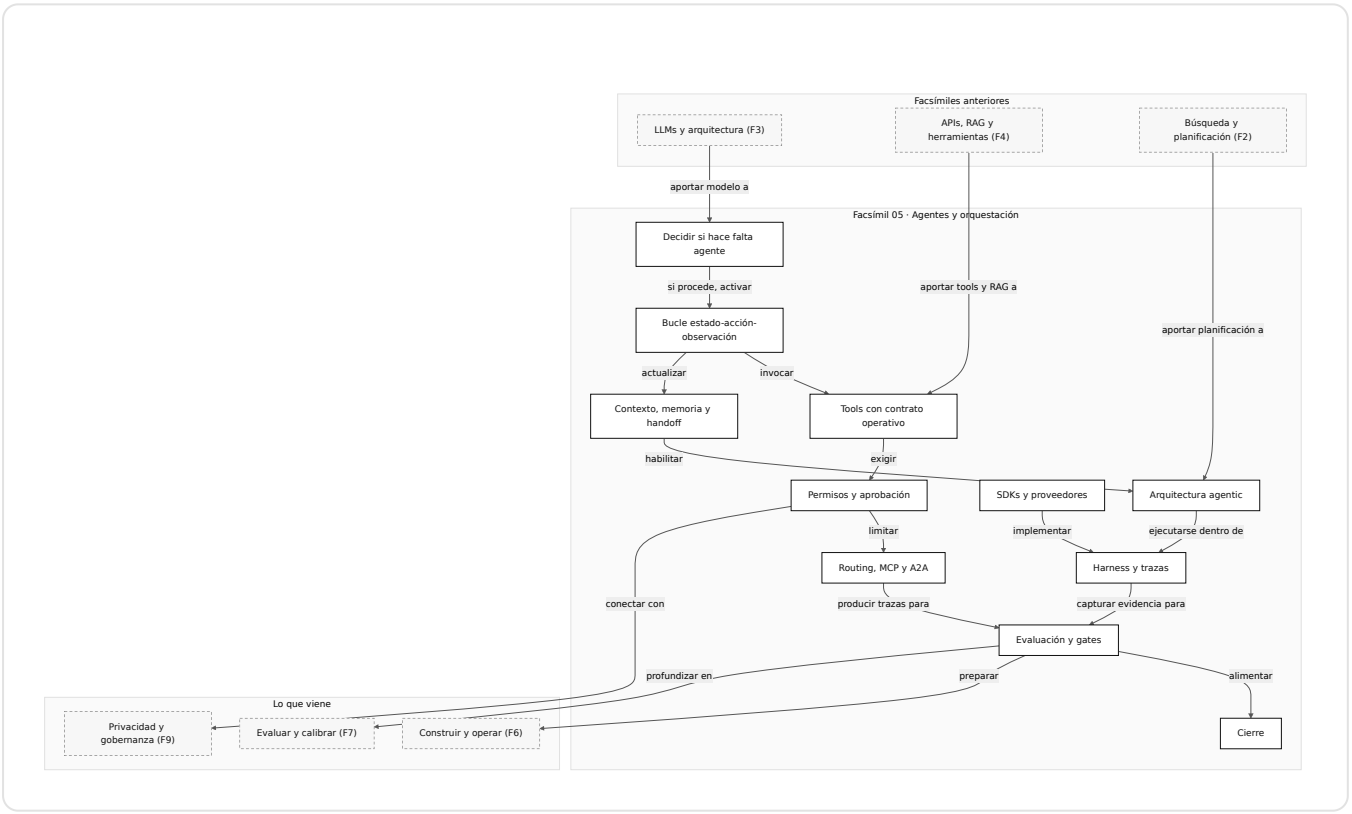
Visto así, cada capítulo no fue un tema suelto, sino una pieza de la misma respuesta a una pregunta vieja de la inteligencia artificial: cómo decidir bien bajo incertidumbre. Un agente es un proceso de decisión sobre observaciones parciales, un POMDP, y todo lo que hemos construido fue darle forma operable a ese formalismo. El estado abstracto se convirtió en contexto y memoria que sabemos mantener y caducar. Las acciones se convirtieron en tools con contrato, precondición y permiso. Las observaciones se convirtieron en salidas tipadas de las que el agente puede fiarse. La recompensa se convirtió en un gate de evaluación que distingue el acierto sólido del frágil. Y los límites del problema se convirtieron en presupuesto, permisos graduados y trazas. No inventamos nada nuevo bajo el sol; le pusimos arnés de ingeniería a una idea que la IA estudia desde hace medio siglo.

De ese recorrido sale un criterio que vale más que cualquier framework concreto, porque los frameworks cambiarán y el criterio no. Ante cualquier sistema que «use IA para hacer cosas», sabes ahora qué preguntar: ¿qué problema resuelve y necesita de verdad un bucle, o le bastaba un prompt o una tool? ¿Qué ve como estado y qué arrastra como contexto sin querer? ¿Qué puede hacer cada tool y quién autoriza las acciones con efecto? ¿Cómo se mide su éxito, mirando la trayectoria y no solo la respuesta? ¿Cuándo para, y puede explicar por qué? Quien sabe hacer esas preguntas no necesita que le digan si un agente está bien construido: lo ve. Y quien construye sabiendo que se las van a hacer, construye distinto desde el primer día.

El facsímil siguiente parte justo de aquí. Coordinar herramientas con criterio era el final de este; coordinar agentes, gobernar su operación y medir su impacto será el principio del próximo. Pero la base no cambia: un sistema más capaz no es uno con más autonomía, sino uno cuya autonomía está mejor diseñada. Esa frase, que al empezar el facsímil sonaba a eslogan, ahora debería sonar a método.



Cómo encaja todo



El mapa ya no resume capítulos: muestra una arquitectura de sistema. La parte superior obliga a justificar requisitos, ADR, SLO, contratos y dataset de aceptación. La zona central modela el agente como runtime con estado, cola, idempotencia, router, policy engine, memoria, dispatcher y sistemas externos. La parte inferior aterriza lo que un equipo de ingeniería necesita para operar: trazas, métricas, evals, gates, CI/CD, configuración versionada y auditoría.

Vocabulario aprendido

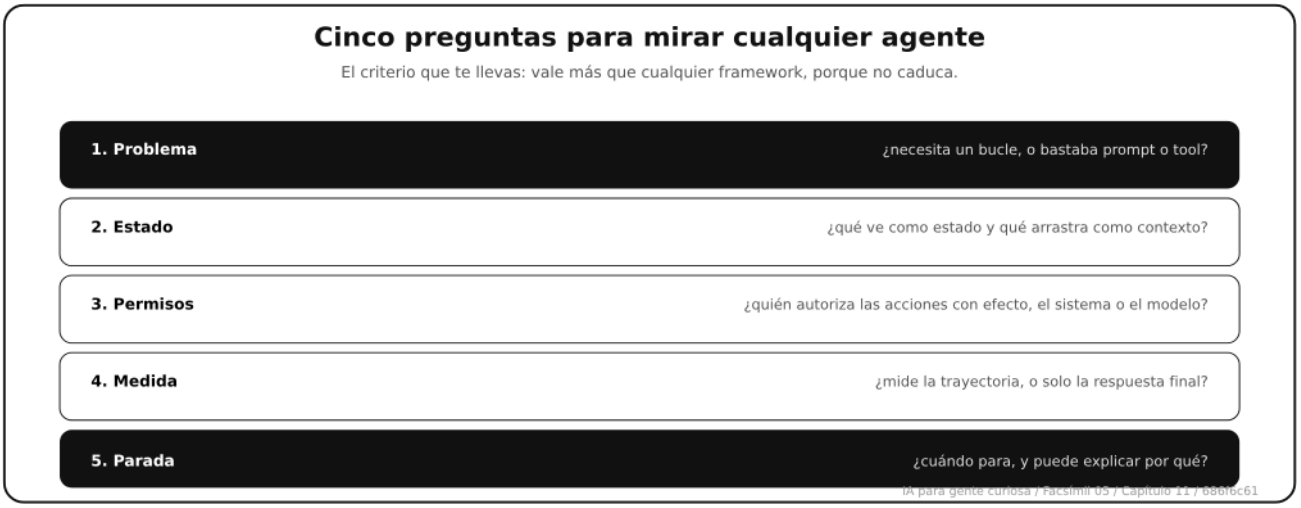
TÉRMINO	DEFINICIÓN ÚTIL
Agente operable	Sistema agentic que se puede limitar, observar, reproducir y evaluar.
Tool contract	Descripción verificable de argumentos, semántica, efecto, errores y permisos.
Harness	Entorno que ejecuta el agente con límites, sensores, trazas y fixtures.
Handoff	Paso explícito de contexto y responsabilidad entre componentes o agentes.
MCP	Protocolo para exponer tools y contexto a modelos o agentes mediante contrato externo.
A2A	Patrón/protocolo para que un agente o sistema delegue trabajo en otro que también decide.
Approval gate	Punto donde una persona debe revisar una acción antes de que tenga efecto.
Trace grading	Evaluación estructurada de una traza, no solo de una respuesta final.
Baseline	Versión de referencia contra la que comparas una candidata.
Regresión	Empeoramiento de un comportamiento que antes funcionaba.
Idempotencia	Propiedad por la que repetir una petición no duplica el efecto.
Backpressure	Mecanismo para frenar entrada cuando el sistema no puede procesar más sin degradarse.
SLO	Objetivo operativo medible, como p95 de latencia o coste máximo por run aceptada.
ADR	Registro breve de una decisión de arquitectura y sus razones.
Trace context	Información propagada para correlacionar spans de una misma ejecución.

El Model Context Protocol define una forma estándar de conectar aplicaciones de IA con tools y datos externos.[12] A2A, por su parte, se orienta a comunicación entre agentes o sistemas agentic con tareas, mensajes y artefactos.[13]

La evaluación completa cierra el círculo. OpenAI Agents SDK organiza las ejecuciones en trazas y spans que permiten inspeccionar llamadas de modelo, tools, guardrails y handoffs.[14] Google ADK separa evaluación de trayectoria/tool use y respuesta final.[15] Promptfoo plantea los agentes de código como sistemas que deciden, actúan, observan y repiten, por lo que recomienda assertions de ruta, coste, latencia, permisos y repetición.[16] AgentBench refuerza la misma idea desde benchmark académico: evaluar agentes exige entornos interactivos y no solo preguntas de texto.[17]

Dónde solía tropezar yo

TROPIEZO	POR QUÉ PASA	ANTÍDOTO
Llamar agente a cualquier chat	La palabra suena avanzada y vende bien.	Preguntar por estado, tools, acciones, observaciones y evaluación.
Diseñar tools sin pensar efectos	El schema parece suficiente.	Añadir semántica, permisos, idempotencia, errores y traza.
Guardar memoria sin política	Parece útil recordarlo todo.	Definir qué se guarda, por cuánto tiempo, quién lo ve y cómo se borra.
Elegir SDK antes de diseñar arquitectura	El proveedor da una sensación de camino hecho.	Escribir primero el contrato de run, tools, permisos y evaluación.
Añadir subagentes demasiado pronto	Multiplica rutas y fallos antes de medir.	Empezar con workflow simple y subir complejidad solo si hay evidencia.
Evaluar solo la respuesta final	Es lo que se ve en pantalla.	Medir trayectoria, coste, latencia, permisos y trazas.
No cerrar el bucle de regresiones	El mismo fallo vuelve con otro nombre.	Cada fallo importante crea caso permanente en el dataset.
No modelar estados	El agente parece flexible, pero nadie sabe dónde puede quedarse atascado.	Dibujar estados, transiciones válidas y salidas de error.
Olvidar idempotencia	Un reintento puede duplicar una acción persistente.	Usar `idempotency_key` y registrar efectos antes de repetir.
No definir SLO	Todo parece aceptable hasta que hay usuarios reales.	Fijar p95, coste, tasa de fallo y presupuesto por ruta.



Antes de pasar página

Responde estas preguntas sin mirar el texto. Si alguna se te escapa, vuelve al capítulo indicado.

PREGUNTA	VUELVE A
¿Cuándo basta un prompt y cuándo aparece un agente?	Capítulo 01.
¿Puedes explicar estado, acción, observación y política con un ejemplo?	Capítulo 02.
¿Qué debe contener una tool para ser operable?	Capítulo 03.
¿Qué diferencia hay entre contexto, memoria, compaction y handoff?	Capítulo 04.
¿Qué arquitectura agentic elegirías para una tarea multi-paso y por qué?	Capítulo 05.
¿Cómo harías reproducible una ejecución?	Capítulo 06.
¿Qué no deberías delegar al SDK?	Capítulo 07.
¿Qué acciones requieren aprobación humana?	Capítulo 08.
¿Cuándo usarías MCP, A2A o un workflow local?	Capítulo 09.
¿Cómo decidirías si una versión nueva puede publicarse?	Capítulo 10.
¿Puedes dibujar la máquina de estados de una run?	`Lo que faltaría si lo revisa un ingeniero informático`.
¿Qué harías para que una petición repetida no duplique efectos?	`Lo que faltaría si lo revisa un ingeniero informático`.
¿Qué SLO usarías para operar el agente?	`Lo que faltaría si lo revisa un ingeniero informático`.

En resumen

IDEA FUERZA	QUÉ TE LLEVAS
Un agente es arquitectura, no etiqueta.	Debe tener bucle, estado, acciones, observaciones y control.
Las tools son contratos, no simples funciones.	Importan argumentos, errores, efectos, permisos y trazas.
La memoria exige gobierno.	Guardar contexto sin política puede empeorar calidad, privacidad y coste.
La orquestación reparte responsabilidad.	Routing, MCP, A2A y handoffs deben decir quién decide y qué viaja.
La autonomía se gradúa.	No todo debe ejecutarse sin aprobación ni todo necesita revisión manual.
La evaluación mira trayectoria.	Una respuesta final buena no basta si el camino fue caro, opaco o fuera de contrato.
La ingeniería aparece en los bordes.	Idempotencia, colas, timeouts, SLO, versionado y trazas separan producto de demo.
El criterio se demuestra construyendo.	Construir, medir y justificar es la forma de comprobar que entendimos.

Recursos para seguir: leer, construir y experimentar

Un agente es un modelo con herramientas, estado y una frontera de permisos. Aquí tienes por dónde seguir, sin perder de vista que la decisión de actuar vive fuera del modelo.

Para experimentar sin código. En la caja «Pruébalo en 5 minutos» del capítulo de function calling viste lo esencial: en un playground (`platform.openai.com` , `aistudio.google.com` , `console.anthropic.com`) puedes declarar una herramienta desde la interfaz y ver cómo el modelo, en vez de inventarse el dato, emite una llamada estructurada con argumentos. Ese es el ladrillo de cualquier agente.

Para construir. Cuando quieras montar un agente de verdad, los SDK de referencia son el Agents SDK de OpenAI, el de Anthropic y el ADK de Google; para grafos de control más complejos, LangGraph; y para conectar herramientas y servicios de forma estándar, el Model Context Protocol (MCP). La regla del facsículo se mantiene: el modelo propone, tu política decide, y las acciones sensibles pasan por aprobación.

Para leer. La idea que articula los agentes modernos es el patrón ReAct (razonar y actuar en bucle); cada capítulo enlaza sus referencias en «Para saber más». Lo que junta tools, memoria, permisos y trazas en un agente operable es justamente el salto entre «el modelo puede llamar tools» y «este agente se puede desplegar con control».

Para saber más

Google. (2026). *Agent Development Kit: Evaluate*. https://google.github.io/adk-docs/evaluate/

Google. (2026). *Agent2Agent Protocol Specification*. https://a2a-protocol.org/latest/specification/

Amershi, S. y otros (2019). *Software Engineering for Machine Learning: A Case Study*. *International Conference on Software Engineering: Software Engineering in Practice*, 291-300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

Baylor, D. y otros (2017). *TFX: A TensorFlow-Based Production-Scale Machine Learning Platform*. *Proceedings of KDD*, 1387-1395. https://doi.org/10.1145/3097983.3098021

Jennings, N. R., Sycara, K. y Wooldridge, M. (1998). *A Roadmap of Agent Research and Development*. *Autonomous Agents and Multi-Agent Systems*, 1(1), 7-38. https://doi.org/10.1023/A:1010090405266

Liu, X. y otros (2024). *AgentBench: Evaluating LLMs as Agents*. *International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.2308.03688

Model Context Protocol. (2026). *Specification*. https://modelcontextprotocol.io/specification

OpenAI. (2026). *Agents SDK: Tracing*. https://openai.github.io/openai-agents-python/tracing/

OpenTelemetry. (2026). *Tracing API*. https://opentelemetry.io/docs/specs/otel/trace/api/

Promptfoo. (2026). *Evaluate Coding Agents*. https://www.promptfoo.dev/docs/guides/evaluate-coding-agents/

Preston-Werner, T. (2026). *Semantic Versioning 2.0.0*. https://semver.org/

Qin, Y. y otros (2023). *ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs*. https://doi.org/10.48550/arXiv.2307.16789

Sculley, D. y otros (2015). *Hidden Technical Debt in Machine Learning Systems*. *Advances in Neural Information Processing Systems*. https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems

Smith, R. G. (1980). *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*. *IEEE Transactions on Computers*, C-29(12), 1104-1113. https://doi.org/10.1109/TC.1980.1675516

W3C. (2021). *Trace Context*. https://www.w3.org/TR/trace-context/

Wooldridge, M. y Jennings, N. R. (1995). *Intelligent Agents: Theory and Practice*. *The Knowledge Engineering Review*, 10(2), 115-152. https://doi.org/10.1017/S0269888900008122

Yao, S. y otros (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*. *International Conference on Learning Representations*. https://arxiv.org/abs/2210.03629

Cuadernos para practicar

Has coordinado tools, memoria y permisos a lo largo del facsímil; estos cuadernos te dejan abrir el motor de un agente. Son *notebooks* que se abren en Google Colab —gratis, en el navegador— o te puedes descargar. Cada uno, explicado paso a paso, con salidas reales, y anclado al capítulo del que sale.

Un agente ReAct mínimo, paso a paso

Qué practicas: construir el bucle pensar-actuar-observar de un agente con herramientas reales. **Dónde encaja:** capítulos 2, 3 y 5 (estado/acción/observación; *tools*; de ReAct a multiagente). **Qué necesitas:** un navegador. Corre en CPU; sin claves (el «modelo» se simula con reglas).

Construyes el esqueleto de un agente que descompone una tarea que no se contesta de un tirón —«quiero 3 teclados, cuánto es y cómo va mi pedido 1002»— en pasos: busca el precio en el catálogo, calcula el total (**89,70 €**), consulta el estado del pedido y solo entonces responde. Ves su traza razonando en voz alta (pensamiento, acción, observación) en cada vuelta. Y compruebas por qué importa el **tope de pasos**: con un cerebro roto, el bucle se corta a los 5 pasos en vez de girar para siempre. La decisión de qué herramienta usar la simulamos con reglas; en un agente real la toma el LLM, pero el bucle es idéntico.

Un agente no es un prompt más listo: es un bucle alrededor del modelo. Entenderlo es entender por qué encadena pasos... y por qué a veces se va por las ramas.

► Abrir en Google Colab **↓ Descargar el cuaderno (.ipynb)**

Function calling: el contrato entre el modelo y tus herramientas

Qué practicas: validar lo que el modelo «pide» ejecutar antes de tocar nada. **Dónde encaja:** capítulo 3 (*tools* y contratos operativos: *function calling*). **Qué necesitas:** un navegador. Corre en CPU; sin claves (se simula lo que pide el modelo).

Ves que un modelo no ejecuta tus funciones: rellena un formulario —nombre de la herramienta y argumentos— y tu código valida antes de disponer. Defines dos herramientas con su contrato (convertir moneda, crear recordatorio) y dejas pasar las llamadas bien hechas: 100 USD se convierten a 92 EUR, el recordatorio se crea. Luego llegan las alucinaciones y se quedan todas en la puerta con un motivo legible: una moneda inventada (BTC), unos minutos negativos, un campo que falta, una herramienta que no existe. Ninguna toca nada.

Entre el modelo y tu base de datos hay una frontera. *Function calling* es esa frontera: el modelo propone, tu código valida y dispone.

► **Abrir en Google Colab** ↓ **Descargar el cuaderno (.ipynb)**

Un mini auto-scheduler: el bucle con verificador

Qué practicas: el patrón propón-verifica-mide-refina, el corazón de los agentes que optimizan algo medible. **Dónde encaja:** capítulo 11 (el bucle con verificador: el LLM que propone y el entorno que mide). **Qué necesitas:** un navegador. Corre en CPU; sin claves (la «política» que propone se simula con búsqueda).

Construyes un **auto-scheduler** que acelera una operación real (multiplicar matrices) probando transformaciones —orden de bucles, tamaño de bloque— y quedándose con la mejor. La gracia es el **verificador**: ejecuta cada variante, **comprueba que el resultado es correcto** (si no, la transformación es ilegal y se rechaza) y **mide el tiempo de verdad**. Un bucle propone, el verificador juzga, y el mejor va subiendo iteración a iteración. Ves un caso rechazado por incorrecto, compruebas que **best-of-N** supera a una sola tirada y cierras con la media geométrica de speedups. Cambiar esa política de búsqueda por un LLM que lea el feedback es, exactamente, lo que hace COMPILOT.

No es un agente que «suena convincente»: es un agente cuya observación es una señal dura. Ese anclaje es lo que separa optimizar de alucinar.

► **Abrir en Google Colab** ↓ **Descargar el cuaderno (.ipynb)**

Notas

1. Wooldridge, M. y Jennings, N. R. (1995). *Intelligent Agents: Theory and Practice*. The Knowledge Engineering Review, 10(2), 115-152. https://doi.org/10.1017/S0269888900008122. Consultado el 10 de junio de 2026. ↩

2. OpenTelemetry. (2026). *Tracing API*. https://opentelemetry.io/docs/specs/otel/trace/api/. Consultado el 10 de junio de 2026. ↩

3. W3C. (2021). *Trace Context*. https://www.w3.org/TR/trace-context/. Consultado el 10 de junio de 2026. ↩

4. Preston-Werner, T. (2026). *Semantic Versioning 2.0.0*. https://semver.org/. Consultado el 10 de junio de 2026. ↩

5. Sculley, D. y otros (2015). *Hidden Technical Debt in Machine Learning Systems*. Advances in Neural Information Processing Systems. https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems. Consultado el 10 de junio de 2026. ↩

6. Amershi, S. y otros (2019). *Software Engineering for Machine Learning: A Case Study*. International Conference on Software Engineering: Software Engineering in Practice, 291-300. https://doi.org/10.1109/ICSE-SEIP.2019.00042. Consultado el 10 de junio de 2026. ↩

7. Baylor, D. y otros (2017). *TFX: A TensorFlow-Based Production-Scale Machine Learning Platform*. Proceedings of KDD, 1387-1395. https://doi.org/10.1145/3097983.3098021. Consultado el 10 de junio de 2026. ↩

8. Yao, S. y otros (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*. International Conference on Learning Representations. https://arxiv.org/abs/2210.03629. Consultado el 10 de junio de 2026. ↩

9. Qin, Y. y otros (2023). *ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs*. https://doi.org/10.48550/arXiv.2307.16789. Consultado el 10 de junio de 2026. ↩

10. Smith, R. G. (1980). *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*. IEEE Transactions on Computers, C-29(12), 1104-1113. https://doi.org/10.1109/TC.1980.1675516. Consultado el 10 de junio de 2026. ↩

11. Jennings, N. R., Sycara, K. y Wooldridge, M. (1998). *A Roadmap of Agent Research and Development*. Autonomous Agents and Multi-Agent Systems, 1(1), 7-38. https://doi.org/10.1023/A:1010090405266. Consultado el 10 de junio de 2026. ↩

12. Model Context Protocol. (2026). *Specification*. https://modelcontextprotocol.io/specification. Consultado el 10 de junio de 2026. ↩

13. Google. (2026). *Agent2Agent Protocol Specification*. https://a2a-protocol.org/latest/specification/. Consultado el 10 de junio de 2026. ↩

.4. OpenAI. (2026). *Agents SDK: Tracing*. https://openai.github.io/openai-agents-python/tracing/. Consultado el 10 de junio de 2026. ↩

.5. Google. (2026). *Agent Development Kit: Evaluation*. https://google.github.io/adk-docs/evaluate/. Consultado el 10 de junio de 2026. ↩

.6. Promptfoo. (2026). *Evaluate Coding Agents*. https://www.promptfoo.dev/docs/guides/evaluate-coding-agents/. Consultado el 10 de junio de 2026. ↩

.7. Liu, X. y otros (2024). *AgentBench: Evaluating LLMs as Agents*. International Conference on Learning Representations. https://doi.org/10.48550/arXiv.2308.03688. Consultado el 10 de junio de 2026. ↩

.8. Sutton, R. S. y Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2.ª ed.). MIT Press. La función de transición define cómo evoluciona el estado de un proceso de decisión secuencial. ↩

.9. Fielding, R., Nottingham, M. y Reschke, J. (2022). *HTTP Semantics* (RFC 9110). https://datatracker.ietf.org/doc/html/rfc9110 Define método idempotente: el efecto pretendido es el mismo se ejecute una o varias veces. ↩