Facsímil 05

Agentes y orquestación

Capítulo 09

Orquestación: routing, MCP, A2A y ADKs

686f6c61 · 2026

Capítulo 09: Orquestación: routing, MCP, A2A y ADKs

Cuando un agente deja de estar solo

Hasta ahora hemos construido piezas: qué es un agente, cómo usa tools, cómo conserva contexto, qué arquitecturas existen, qué SDKs hay y cómo se revisan acciones con impacto. Pero un sistema real rara vez vive con un solo agente y una sola herramienta.

En el capítulo 03 definimos tools con contrato. En el capítulo 04 vimos contexto, memoria y handoff. En el capítulo 05 separamos arquitecturas de agentes. En el capítulo 07 entramos en SDKs y ADKs. Y en el capítulo 08 pusimos permisos alrededor de todo eso.

En cuanto aparece una aplicación seria, llegan preguntas nuevas: ¿uso una tool local o un servidor MCP?, ¿delego a otro agente?, ¿hago routing por coste, por latencia, por especialidad o por permisos?, ¿qué pasa si el primer proveedor falla?, ¿cómo sé qué agente sabe hacer qué?, ¿cómo evito que la lista de herramientas llene el contexto?, ¿dónde guardo la traza para comparar rutas?

Este capítulo va de esa capa: **orquestar**. Orquestar no es poner más agentes. Es decidir, con criterio verificable, qué pieza ejecuta cada parte del trabajo.

Qué no es orquestar

Orquestar no es encadenar diez llamadas al modelo y esperar que el resultado parezca inteligente. Eso suele producir latencia, coste y poca trazabilidad.

Tampoco es esconder todas las decisiones dentro de un prompt del tipo “elige la mejor herramienta”. A veces el modelo debe elegir. Otras veces la decisión debe ser una regla de negocio, un routing por permisos, un gate de coste, un clasificador pequeño o una tabla de capacidades mantenida por ingeniería.

Y no es confundir protocolos. MCP no convierte una herramienta en un agente completo. A2A no sustituye un contrato de tool. Un ADK no elimina la necesidad de decidir qué estado guardas, qué permiso aplicas y qué métrica vas a mirar después.

La definición útil

Para este facsímil, orquestación agentic es:

La capa que convierte una intención en un plan de ejecución trazable: selecciona ruta, herramienta, agente, protocolo, modelo, permisos y estrategia de reintento antes de ejecutar.

La orquestación es, en el fondo, una **función de enrutamiento** (un dispatcher): toma la petición y el estado y decide a qué destino va, con qué contrato, permisos y traza. En notación de función:

$$o(q, s, C, P, B) \to \langle ruta, contrato, permisos, traza \rangle$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
o	Función de orquestación.	El componente que decide si usar una tool local, MCP, A2A o una cola humana.
q	Petición actual.	“Revisa esta cita y actualiza la bibliografía”.
s	Estado de la ejecución.	Usuario, sesión, historial, pasos previos, errores, coste acumulado.
C	Catálogo de capacidades.	Qué sabe hacer cada tool, agente o servicio.
P	Política de permisos.	Qué rutas puede usar este usuario en este entorno.
B	Presupuesto.	Latencia máxima, coste máximo, tokens, número de tools, retries.
$\langle ruta, contrato, permisos, traza \rangle$	Salida de orquestación.	“usar `mcp_biblioteca.search_paper`, con aprobación si escribe, registrar `trace_id`”.

La definición parece formal, pero la idea es sencilla: antes de ejecutar, el sistema debe saber **por qué esa ruta y no otra**.

Fecha de corte del estado del arte

Fecha de corte: 10 de junio de 2026.
Fuentes consultadas ese día: especificación pública de MCP, documentación de OpenAI Agents SDK sobre MCP y handoffs, documentación de Anthropic sobre MCP connector, documentación de Google ADK sobre MCP tools, A2A, routing de agentes, routing de modelos y workflow agents, especificación A2A, y referencias clásicas de sistemas multiagente.

Lo estable es el patrón: separar capacidades, contratos, permisos, routing, ejecución y trazas. Lo cambiante son nombres de clases, transportes soportados, versiones beta, conectores hospedados, compatibilidad por proveedor y gobernanza de protocolos.

De dónde viene esta idea

Los agentes no nacieron con los LLMs. Wooldridge y Jennings ya definían agentes por propiedades como autonomía, reactividad, proactividad y habilidad social: un agente no solo calcula; actúa en un entorno y se coordina con otros.[1]

Jennings, Sycara y Wooldridge describieron el campo multiagente como una forma de repartir control, conocimiento y capacidad de acción entre entidades que cooperan para resolver tareas.[2] Y mucho antes de hablar de LLMs, Smith propuso Contract Net Protocol: un mecanismo donde un coordinador anuncia tareas, otros componentes proponen cómo resolverlas y se asigna el trabajo según criterios.[3]

No copiamos esos protocolos clásicos sin más. Pero nos sirven para una idea importante: **la delegación necesita contrato**. Si alguien va a recibir una tarea, debe declarar qué sabe hacer, qué necesita, qué devuelve, cuánto tarda, cuánto cuesta y cómo falla.

Las tripas de la orquestación

Una arquitectura de orquestación publicable suele tener estas piezas:

PIEZA	QUÉ DECIDE	QUÉ DEBERÍA REGISTRAR
Router	Qué ruta intenta primero.	Señales usadas, alternativas descartadas y motivo.
Capability registry	Qué capacidades existen.	Versión, dueño, coste, latencia, permisos, contrato.
Tool gateway	Cómo se ejecutan tools.	Input validado, output, errores, duración, efecto.
MCP client	Cómo se conectan servidores MCP.	Servidor, tools listadas, auth, tool llamada, resultado.
A2A client	Cómo se invocan agentes externos.	AgentCard, Task, mensajes, artefactos, estado.
Policy engine	Qué se permite o revisa.	Decisión, scope, persona revisora si aplica.
Run state	Qué está pasando ahora.	Paso, ruta activa, retries, presupuesto restante.
Trace store	Qué ocurrió realmente.	Eventos, spans, costes, latencias, decisiones.
Eval harness	Qué ruta fue mejor.	Tasa de acierto, coste, latencia, reintentos, calidad.

El router no debería ser un oráculo. Puede ser una función pequeña, una regla, un clasificador, un modelo barato, un grafo o una mezcla. Lo importante es que sus decisiones se puedan revisar.

Fórmula práctica para elegir ruta

Una forma útil de pensar el routing es puntuar cada ruta con una función de valor aditiva, la decisión multicriterio de siempre: sumar las señales a favor y restar las penalizaciones.[4] No es para convertirlo en matemática falsa, sino para obligarnos a nombrar las señales. Y cuando hay datos, esa puntuación puede **aprenderse** en lugar de fijarse a mano, como en los enrutadores de modelos.[5]

$$R_i = \alpha S_i + \beta Q_i + \gamma A_i - \delta L_i - \epsilon C_i - \zeta K_i$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
R_i	Puntuación de la ruta i.	Ruta `mcp_biblioteca` obtiene 0,74.
S_i	Encaje semántico con la petición.	La ruta sabe buscar referencias: 0,90.
Q_i	Calidad esperada o histórica.	En evaluaciones acertó 84%: 0,84.
A_i	Disponibilidad actual.	Servicio sano: 1,00.
L_i	Latencia normalizada.	1,8 s sobre máximo 5 s: 0,36.
C_i	Coste normalizado.	0,03 euros sobre máximo 0,10: 0,30.
K_i	Riesgo operativo normalizado.	Escritura externa sin revisión: 0,70.
$\alpha, \beta, \gamma, \delta, \epsilon, \zeta$	Pesos de decisión.	Dar más peso a calidad que a coste en tareas críticas.

Ejemplo numérico:

SEÑAL	RUTA LOCAL	RUTA MCP	RUTA A2A
S_i	0,40	0,92	0,80
Q_i	0,70	0,86	0,88
A_i	1,00	0,95	0,90
L_i	0,10	0,35	0,55
C_i	0,05	0,25	0,40
K_i	0,10	0,35	0,45

Con pesos $\alpha = 0{,}30$, $\beta = 0{,}25$, $\gamma = 0{,}15$, $\delta = 0{,}10$, $\epsilon = 0{,}10$, $\zeta = 0{,}10$:

RUTA	CÁLCULO	RESULTADO
Local	$0{,}30 \cdot 0{,}40 + 0{,}25 \cdot 0{,}70 + 0{,}15 \cdot 1 - 0{,}10 \cdot 0{,}10 - 0{,}10 \cdot 0{,}05 - 0{,}10 \cdot 0{,}10$	0,42
MCP	$0{,}30 \cdot 0{,}92 + 0{,}25 \cdot 0{,}86 + 0{,}15 \cdot 0{,}95 - 0{,}10 \cdot 0{,}35 - 0{,}10 \cdot 0{,}25 - 0{,}10 \cdot 0{,}35$	0,54
A2A	$0{,}30 \cdot 0{,}80 + 0{,}25 \cdot 0{,}88 + 0{,}15 \cdot 0{,}90 - 0{,}10 \cdot 0{,}55 - 0{,}10 \cdot 0{,}40 - 0{,}10 \cdot 0{,}45$	0,46

La ruta MCP gana. Pero la decisión final todavía debe pasar por permisos. Si esa ruta escribe, publica o consulta datos sensibles, el score no basta.

Routing: reglas, modelo o grafo

Hay tres familias de routing que conviene distinguir.

La primera es routing determinista. Si la petición contiene un pago, va al flujo de pagos. Si pide una cita bibliográfica, va al agente de referencias. Si modifica producción, pide revisión. Es simple, barato y fácil de auditar.

La segunda es routing por clasificación. Un clasificador ligero, que puede ser un modelo pequeño o una función entrenada, decide si la tarea es simple, compleja, técnica, legal, de datos, de escritura o de soporte. Google ADK documenta `RoutedAgent` para elegir un agente por invocación, con fallback si el agente seleccionado falla antes de emitir eventos.[6] También documenta `RoutedLlm` para elegir entre modelos cuando solo cambia el modelo y no cambian instrucciones, tools o subagentes.[7]

La tercera es routing por workflow o grafo. Aquí no elegimos un único destino, sino un recorrido: primero recuperar contexto, luego resolver, después verificar, y finalmente decidir si publicar o pedir revisión. En ADK, los workflow agents ejecutan patrones secuenciales, paralelos o de bucle con lógica predefinida; la propia documentación indica que en ADK 2.0 los workflows de grafo y dinámicos ofrecen más control y flexibilidad que las plantillas rígidas.[8]

TIPO DE ROUTING	BUENA ELECCIÓN CUANDO...	RIESGO SI SE USA MAL
Regla explícita	La condición es clara y de negocio.	Crece como una lista imposible de mantener.
Clasificador	Hay muchas peticiones parecidas y categorías estables.	Clasifica con seguridad aparente pero sin evidencia.
LLM router	La intención es ambigua y necesita interpretación.	Puede ser caro, lento y difícil de explicar.
Grafo	Hay pasos obligatorios, gates y reintentos.	Se vuelve rígido si cada caso necesita excepción.
Handoff	Hay especialistas con contratos claros.	Se usa para tapar falta de diseño interno.
A2A	El destino es otro sistema agentic independiente.	Se añade protocolo cuando bastaba una tool.

MCP: tools y contexto como contrato externo

MCP, Model Context Protocol, estandariza cómo una aplicación con modelo se conecta a contexto, datos y herramientas. La especificación vigente consultada define hosts, clientes y servidores; usa JSON-RPC 2.0; y organiza capacidades como recursos, prompts y tools.[9]

La frase importante es esta: **MCP no es "un plugin universal"; es una frontera de capacidades**.

CONCEPTO MCP	QUÉ SIGNIFICA	EJEMPLO ENTENDIBLE
Host	Aplicación que usa el modelo.	Un IDE, una app de chat, un panel interno.
Client	Conector dentro del host.	Pieza que habla con un servidor MCP.
Server	Servicio que expone capacidades.	Filesystem, base de datos, calendario, buscador.
Resource	Dato legible o contexto.	`file://capitulo.md`, esquema SQL, documento.
Prompt	Plantilla reutilizable.	"Resume esta incidencia con formato técnico".
Tool	Función ejecutable.	`search_docs`, `read_file`, `create_ticket`.
Capability negotiation	Declaración de lo soportado.	El cliente sabe si hay tools, resources o prompts.
Consentimiento	Revisión de acceso o acción.	La persona autoriza leer una carpeta o llamar una tool.

OpenAI Agents SDK para JavaScript documenta varias formas de usar MCP: tools MCP hospedadas por la Responses API, servidores Streamable HTTP y servidores por `stdio`; también menciona aspectos de ciclo de vida, cache de listado de tools, nombres prefijados por servidor y filtrado de tools.[10]

Anthropic expone MCP desde la Messages API mediante `mcp_servers` y `mcp_toolset`; en la versión consultada requiere beta header `mcp-client-2025-11-20`, soporta tool calls, permite configuración por tool, allowlist, denylist, `defer_loading` y OAuth para servidores remotos.[11]

Google ADK documenta `McpToolset` como mecanismo para integrar tools de servidores MCP: conecta, lista tools mediante `list_tools`, adapta schemas a tools del ADK, expone esas tools al `LlmAgent` y proxifica llamadas mediante `call_tool`; además permite filtrar tools.[12]

El punto de ingeniería: si expones 80 tools MCP a un agente, no has “mejorado” el sistema. Has ampliado el espacio de decisión. Necesitas filtrado, nombres claros, permisos, cache de tool list, límites de coste y evaluación.

A2A: cuando el destino también decide

A2A, Agent2Agent Protocol, no va de que un modelo llame una función. Va de que un sistema agentic hable con otro sistema agentic. Google ADK lo presenta como una forma de construir sistemas multiagente donde agentes distintos colaboran mediante A2A, exponiendo y consumiendo agentes remotos.[13]

La especificación A2A consultada organiza el protocolo alrededor de operaciones como enviar mensajes, enviar mensajes en streaming, obtener tareas, listar tareas, cancelar tareas, suscribirse a una tarea, gestionar notificaciones y obtener una AgentCard extendida.[14]

Los objetos clave son:

OBJETO A2A	QUÉ APORTA	POR QUÉ IMPORTA
`AgentCard`	Identidad, capacidades, skills, interfaces, seguridad y versión.	Permite descubrir qué puede hacer un agente antes de llamarlo.
`AgentSkill`	Capacidad concreta declarada por el agente.	Evita delegar tareas fuera de especialidad.
`Task`	Unidad durable de trabajo.	Permite seguimiento, estados, streaming y recuperación.
`Message`	Interacción entre cliente y agente.	Conserva turnos de comunicación.
`Part`	Fragmento multimodal o estructurado.	Permite texto, archivos, formularios u otros modos.
`Artifact`	Resultado producido.	Separa conversación de entregables.
`AgentCapabilities`	Streaming, notificaciones, extensiones.	Permite validar si una operación está soportada.
`SecurityScheme`	Requisitos de autenticación.	No todos los agentes son públicos ni equivalentes.

La diferencia con MCP:

PREGUNTA	MCP	A2A
¿Qué conecta?	Un agente o host con herramientas y datos.	Un agente con otro agente o sistema agentic.
¿Unidad principal?	Tool, resource, prompt.	AgentCard, task, message, artifact.
¿Quién decide el trabajo interno?	El host o agente que llama la tool.	El agente remoto puede tener su propio bucle y estado.
¿Cuándo usarlo?	Para exponer capacidades concretas.	Para delegar a un sistema con autonomía propia.
¿Error típico?	Exponer demasiadas tools sin permisos.	Usarlo cuando bastaba una API o tool simple.

Si llamas a `search_contracts(query)`, probablemente es MCP o tool normal. Si preguntas a un agente de compras “gestiona este proceso con tus pasos, estado y outputs”, eso se parece más a A2A.

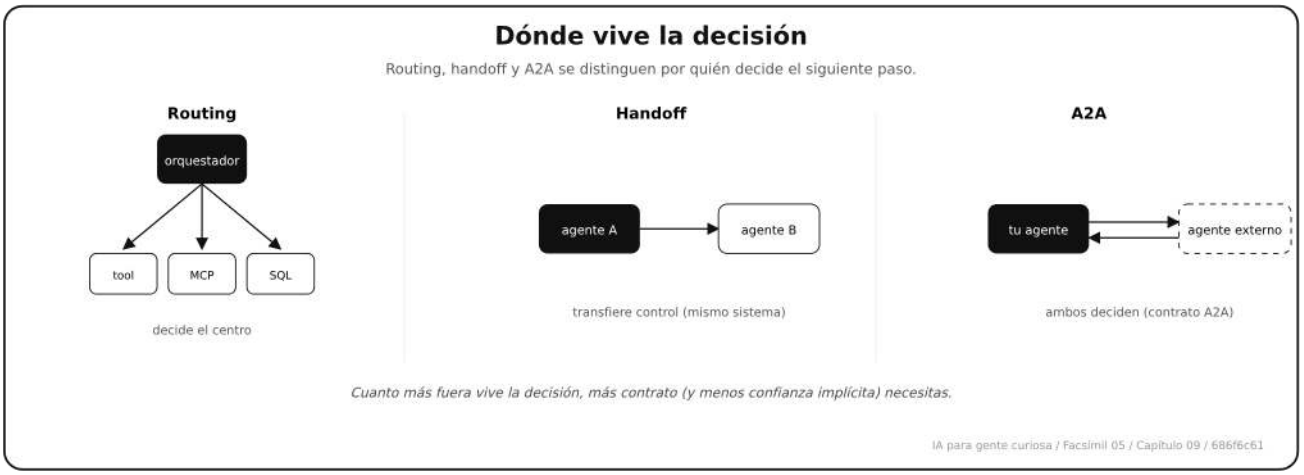
Handoffs, routing y A2A no son lo mismo

En OpenAI Agents SDK, un handoff permite que un agente delegue una tarea a otro agente especializado; se representa como una tool para el LLM, por ejemplo `transfer_to_refund_agent` si existe un agente de devoluciones.[15]

Eso no significa que todo handoff sea A2A. Un handoff dentro de un SDK puede ser interno: mismo proceso, mismo runtime, mismas trazas. A2A aparece cuando el destino es un sistema agentic independiente, con su propia AgentCard, su propio endpoint, sus propias capacidades y su propio ciclo de vida de tareas.

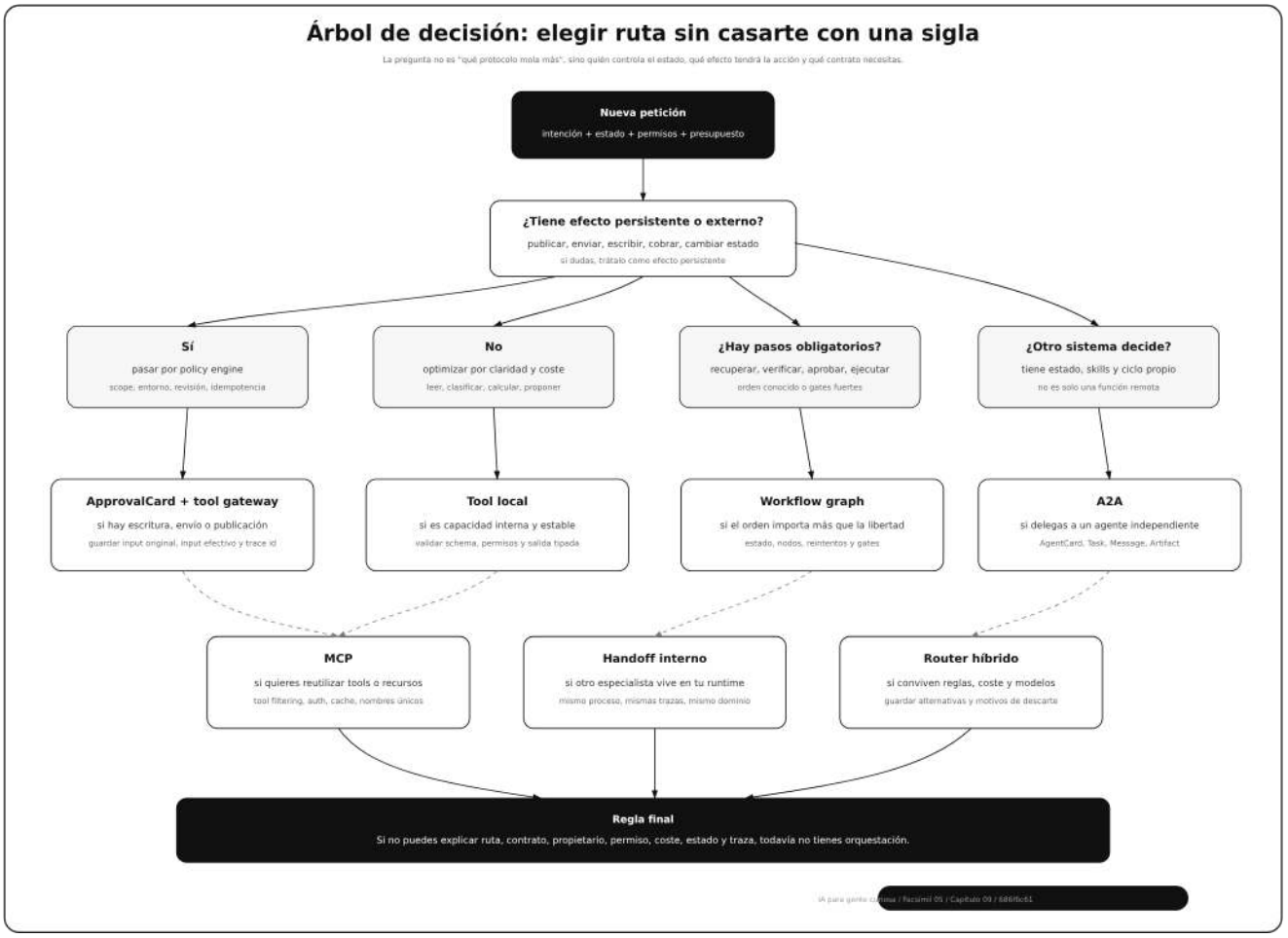
PATRÓN	QUIÉN CONTROLA	UNIDAD DE TRABAJO	CASO TÍPICO
Tool local	Tu aplicación.	Función.	Validar JSON, consultar tabla, calcular score.
MCP server	Tu host y el servidor MCP.	Tool/resource/prompt.	Reutilizar herramientas entre clientes.
Handoff interno	SDK o framework.	Transferencia a especialista.	Agente de soporte deriva a agente técnico.
A2A	Dos sistemas agentic.	Task/mensaje/artefacto.	Tu agente consulta al agente de otra unidad.
Workflow graph	Motor de grafo.	Nodo/estado/transición.	Recuperar, resolver, verificar, publicar.

Estos tres modos no se eligen por gusto, sino por dónde vive la decisión. En el **routing**, un orquestador central decide a qué destino va cada petición (el patrón orquestador-trabajadores).[16] En el **handoff**, el control se transfiere a un especialista dentro de tu propio sistema. En **A2A**, el destino es otro sistema agentic que también decide, así que la frontera deja de ser una función y pasa a ser un contrato entre dos partes.



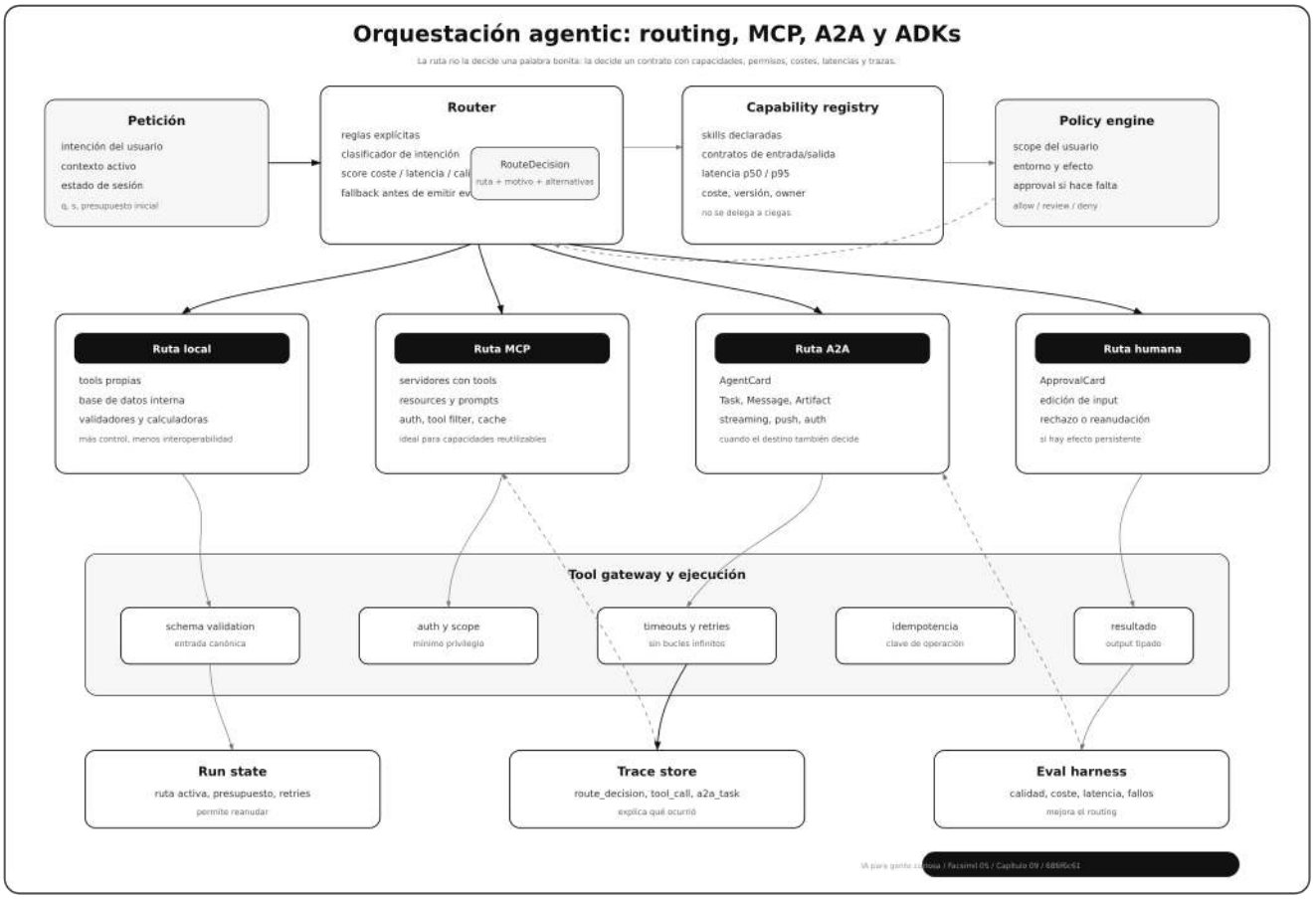
Árbol de decisión para elegir arquitectura

Cuando alguien pregunta “¿uso MCP, A2A, un handoff o una tool?”, yo intentaría que no respondiese desde la moda del momento. Respondería desde el efecto, el propietario, el estado y el contrato.



El árbol obliga a distinguir cuatro preguntas: efecto, control, estado y reutilización. Si una capacidad es interna, estable y de bajo impacto, empieza por tool local. Si quieres reutilizar tools entre clientes, MCP. Si delegas a un especialista dentro del mismo runtime, handoff. Si el destino es un sistema agentic independiente, A2A. Si hay pasos obligatorios, grafo. Si hay efecto persistente, aprobación o policy antes de ejecutar.

Anatomía visual de una orquestación publicable



El diagrama separa algo que en demos suele estar mezclado: el router decide, el registry describe, la policy permite o detiene, el gateway ejecuta y la traza demuestra. MCP y A2A son rutas posibles, no sustitutos de esa arquitectura.

Contratos mínimos: qué viaja realmente

Si alguien solo entiende MCP y A2A como nombres, todavía no puede diseñar bien. Hay que bajar al contrato: qué identificador se envía, qué schema existe, qué estado vuelve, qué permisos intervienen y qué se guarda en la traza.

Los ejemplos siguientes son didácticos. No sustituyen la especificación ni el SDK concreto, pero muestran la forma mental que necesitas para trabajar: capacidades declaradas, argumentos tipados, resultados separados de la conversación y decisiones trazables.

MCP: una tool expuesta por un servidor

Un servidor MCP puede exponer una tool como `search_docs`. Lo importante no es el nombre; es el contrato de entrada y salida. Una tool sin schema obliga al modelo a adivinar.

```
{
  "server_id": "mcp.biblioteca",
  "transport": "streamable_http",
  "tool": {
    "name": "search_docs",
    "description": "Busca documentos normativos y devuelve fragmentos citables.",
    "inputSchema": {
      "type": "object",
      "additionalProperties": false,
      "required": ["query", "limit"],
      "properties": {
        "query": {
          "type": "string",
          "minLength": 4,
          "description": "Pregunta o término de búsqueda."
        },
        "limit": {
          "type": "integer",
          "minimum": 1,
          "maximum": 20,
          "description": "Número máximo de resultados."
        },
        "source_filter": {
          "type": "array",
          "items": { "type": "string" },
          "description": "Colecciones permitidas para esta búsqueda."
        }
      }
    }
  }
}
```

Una llamada a esa tool debería dejar algo así en la traza:

```
{
  "event": "mcp.tool_call",
  "trace_id": "trace-2026-06-10-r1",
  "server_id": "mcp.biblioteca",
  "tool_name": "search_docs",
  "arguments": {
    "query": "normativa permanencia universidad",
    "limit": 5,
    "source_filter": ["normativa_publica"]
  },
  "policy": {
    "decision": "allow",
    "scope": ["read_public"],
    "tool_filter_version": "2026-06-10"
  },
  "result_shape": {
    "documents": "list",
    "citations": "list",
    "elapsed_ms": "number"
  }
}
```

El detalle útil: `tool_filter_version` también se guarda. Si mañana la tool deja de aparecer o cambia de schema, puedes saber con qué catálogo se tomó la decisión.

A2A: AgentCard como expediente de un agente

En A2A no solo quieres saber “hay un agente”. Quieres saber qué skills declara, qué endpoint usa, si soporta streaming, qué seguridad exige y qué versión estás invocando.

```
{
  "name": "Agente de becas",
  "description": "Gestiona revisión inicial de expedientes de becas.",
  "url": "https://becas.example.edu/a2a",
  "version": "1.3.0",
  "capabilities": {
    "streaming": true,
    "pushNotifications": true
  },
  "defaultInputModes": ["text", "application/json"],
  "defaultOutputModes": ["text", "application/json"],
  "skills": [
    {
      "id": "review_scholarship_case",
      "name": "Revisar expediente de beca",
      "description": "Comprueba requisitos, documentación y próximos pasos.",
      "tags": ["becas", "expediente", "revision"],
      "examples": [
        "Revisa el expediente B-1042 con la documentación adjunta."
      ]
    }
  ],
  "securitySchemes": {
    "oauth2": {
      "type": "oauth2",
      "flows": {
        "clientCredentials": {
          "tokenUrl": "https://auth.example.edu/token",
          "scopes": {
            "becas.review": "Permite solicitar revisión de expedientes."
          }
        }
      }
    }
  }
}
```

La `AgentCard` no es marketing. Es parte del contrato operativo. Si no declara capabilities, skills, seguridad y versión, el router está delegando con los ojos cerrados.

A2A: Task como unidad durable de trabajo

Cuando delegas por A2A, no quieres solo un texto de vuelta. Quieres una tarea con estado y artefactos. Eso permite consultar progreso, retomar, cancelar o guardar resultados.

```
{
  "jsonrpc": "2.0",
  "id": "req-2026-06-10-001",
  "method": "message/send",
  "params": {
    "message": {
      "role": "user",
      "parts": [
        {
          "kind": "text",
          "text": "Revisa el expediente B-1042 y devuelve un informe con requisitos
cumplidos, dudas y siguiente paso."
        },
        {
          "kind": "data",
          "data": {
            "case_id": "B-1042",
            "student_scope": "scoped-token-abc",
            "requested_artifact": "decision_report"
          }
        }
      ]
    },
    "metadata": {
      "trace_id": "trace-2026-06-10-r2",
      "caller": "agente-orquestador",
      "budget": {
        "max_latency_ms": 5000,
        "max_cost_eur": 0.10
      }
    }
  }
}
```

Y un resultado razonable debería separar estado, mensaje y artefacto:

```
{
  "task": {
    "id": "task-becas-7781",
    "status": {
      "state": "completed",
      "message": {
        "role": "agent",
        "parts": [
          {
            "kind": "text",
            "text": "Expediente revisado. Falta justificante de residencia."
          }
        ]
      }
    },
    "artifacts": [
      {
        "artifactId": "decision-report-7781",
        "name": "Informe de revisión",
        "parts": [
          {
            "kind": "data",
            "data": {
              "case_id": "B-1042",
              "requirements_ok": ["matricula_activa", "renta_declarada"],
              "missing": ["residencia"],
              "next_step": "Solicitar justificante de residencia antes de continuar."
            }
          }
        ]
      }
    ]
  }
}
```

La separación importa. El mensaje sirve para conversar. El artefacto sirve para operar. Si mezclas ambos, luego no sabes qué parte debe leer una persona y qué parte puede consumir un sistema.

Fallos de producción que conviene ensayar

La orquestación falla de formas bastante previsibles. Lo sensato es probarlas antes de publicar.

FALLO	SEÑAL VISIBLE	CÓMO LO DISEÑARÍA
Tool list desactualizada	El agente intenta llamar una tool que ya no existe.	Cache con versión, invalidación explícita y test de catálogo.
Colisión de nombres	Dos servidores exponen `search` y el modelo elige mal.	Prefijos por servidor y nombres semánticos: `biblioteca.search_docs`.
Schema drift	La tool acepta otros campos o deja de aceptar uno.	Contract tests y validación `additionalProperties: false`.
Permiso heredado de más	Una ruta puede acceder a datos que no necesita.	Scope por tool, usuario, entorno y operación.
Fallback peligroso	Al fallar una ruta, el sistema prueba otra con más impacto.	Fallback solo hacia rutas de igual o menor efecto.
Timeout ambiguo	No sabes si la acción se ejecutó o no.	Idempotency key y consulta de estado antes de reintentar.
Task parcial en A2A	La tarea queda `working` o `input-required`.	Guardar task id, estado y próximo paso; no inventar final.
Coste creciente	Cada ruta añade llamadas de modelo y tool.	Presupuesto por run, cortes por p95 y trazas de coste.
Artefacto perdido	El agente responde, pero no queda entregable usable.	Separar `message` de `artifact` y validar artifact schema.
Ruta no explicable	El resultado es bueno, pero nadie sabe por qué se eligió.	Registrar top candidatos, score, señales y motivo final.

Una prueba sencilla: fuerza cada fallo en entorno de desarrollo. Quita una tool del catálogo. Cambia un schema. Devuelve un timeout. Haz que A2A responda con tarea en progreso. Si el sistema no sabe parar, reintentar o pedir revisión con claridad, todavía no está listo.

Caso para entenderlo: una universidad con sistemas distintos

Imagina una universidad con tres necesidades:

1. Un alumno pregunta por una norma de matrícula.
2. Secretaría necesita consultar expedientes.
3. Un departamento externo tiene su propio agente para becas.

La solución torpe sería dar al agente principal todas las herramientas posibles: calendario, expedientes, normativa, becas, correo, editor, base de datos, CRM y formularios. La solución profesional separa rutas.

PETICIÓN	RUTA PROBABLE	MOTIVO
“¿Qué dice la normativa sobre permanencia?”	MCP documental o RAG interno.	Es consulta de conocimiento cambiante.
“Comprueba si tengo pago pendiente”	Tool interna con permisos.	Afecta datos personales y necesita scope.
“Inicia revisión de beca externa”	A2A con agente de becas.	Otro sistema mantiene estado y proceso propio.
“Redacta respuesta al alumno”	Tool local o agente interno.	Es una propuesta textual sin efecto externo.
“Envía la respuesta oficial”	ApprovalCard antes de tool.	Hay efecto comunicativo y registro institucional.

El mismo usuario puede pasar por varias rutas en una sola tarea. La orquestación no elige “el agente ganador”. Elige el recorrido verificable.

Arquitecturas de orquestación

No existe una única arquitectura correcta. Sí existen patrones reconocibles.

ARQUITECTURA	CÓMO FUNCIONA	CUÁNDO ENCAJA
Router central	Un componente decide cada destino.	Productos con pocas rutas críticas y reglas claras.
Supervisor y especialistas	Un agente supervisor delega a agentes especializados.	Tareas ambiguas donde el modelo puede elegir especialistas.
Grafo de workflow	Nodos y transiciones controlan el recorrido.	Procesos con pasos obligatorios, gates y reintentos.
Tool gateway con MCP	Tools internas y externas se exponen por contratos.	Muchas capacidades reutilizables entre clientes.
A2A federado	Sistemas agentic independientes coordinan tareas.	Varias unidades, proveedores o dominios con autonomía propia.
Router híbrido	Reglas, clasificador, coste, permisos y fallback.	Sistemas en producción con variedad de casos.

Mi recomendación práctica: empezar con router explícito y registry pequeño. Añadir MCP cuando la herramienta deba reutilizarse fuera de un solo agente. Añadir A2A cuando el destino tenga ciclo de vida propio. Añadir grafo cuando hay pasos que no deben quedar a improvisación del modelo.

Decisiones de ingeniería que no se ven en la demo

Una demo puede vivir sin estas piezas. Un sistema serio, no.

DECISIÓN	PREGUNTA QUE DEBES RESPONDER
Versionado de capacidades	¿Qué pasa si `search_docs` cambia su schema mañana?
Nombres únicos	¿Qué ocurre si dos servidores MCP exponen `search`?
Tool filtering	¿Expones todo el servidor o solo tres tools?
Cache de tool list	¿Listas tools en cada run y pagas latencia siempre?
Presupuesto	¿Quién corta una ruta que consume demasiado?
Idempotencia	¿Qué pasa si se reintenta una acción con efecto persistente?
Fallback	¿Cuándo intentas otra ruta y cuándo paras?
Estado parcial	¿Qué haces si A2A devuelve una task en progreso?
Artefactos	¿Dónde guardas archivos, diffs o resultados largos?
Trazas	¿Puedes reconstruir por qué se eligió una ruta?
Evaluación	¿Qué métrica demuestra que el router mejora?

La orquestación es, sobre todo, disciplina de interfaces.

El contrato en la frontera: cuando el destino también decide

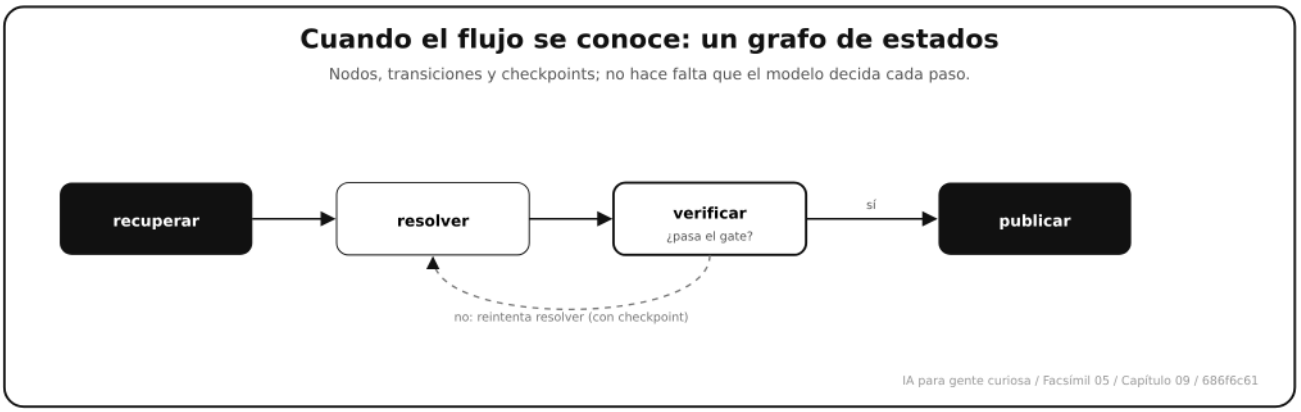
Hay un salto conceptual en este capítulo que merece contarse despacio, porque es donde la orquestación deja de ser un problema de ingeniería interna y se convierte en un problema de relación entre sistemas autónomos. Mientras enrutas hacia tus propias tools, controlas las dos orillas: tú decides la petición y tú implementas el destino. Cuando haces un handoff a un subagente dentro de tu sistema, sigues controlando las dos orillas, aunque el control viaje. Pero en el momento en que llamas a otro sistema agentic que **también decide** (lo que llamamos A2A), pierdes la mitad del control, y eso cambia todo lo que tienes que diseñar.

La diferencia es la misma que separa llamar a una función de llamar a una persona. A una función le pasas argumentos y, si su contrato es determinista, sabes qué te devuelve. A otro agente le pasas una tarea, y ese agente la interpreta, decide cómo abordarla, puede pedir aclaraciones, puede negarse, puede tardar lo que considere y puede devolverte algo que no esperabas, porque por dentro es tan no determinista como el tuyo. La confianza implícita que tienes con tu propio código («sé lo que hace porque lo escribí yo») desaparece. Y lo que ocupa

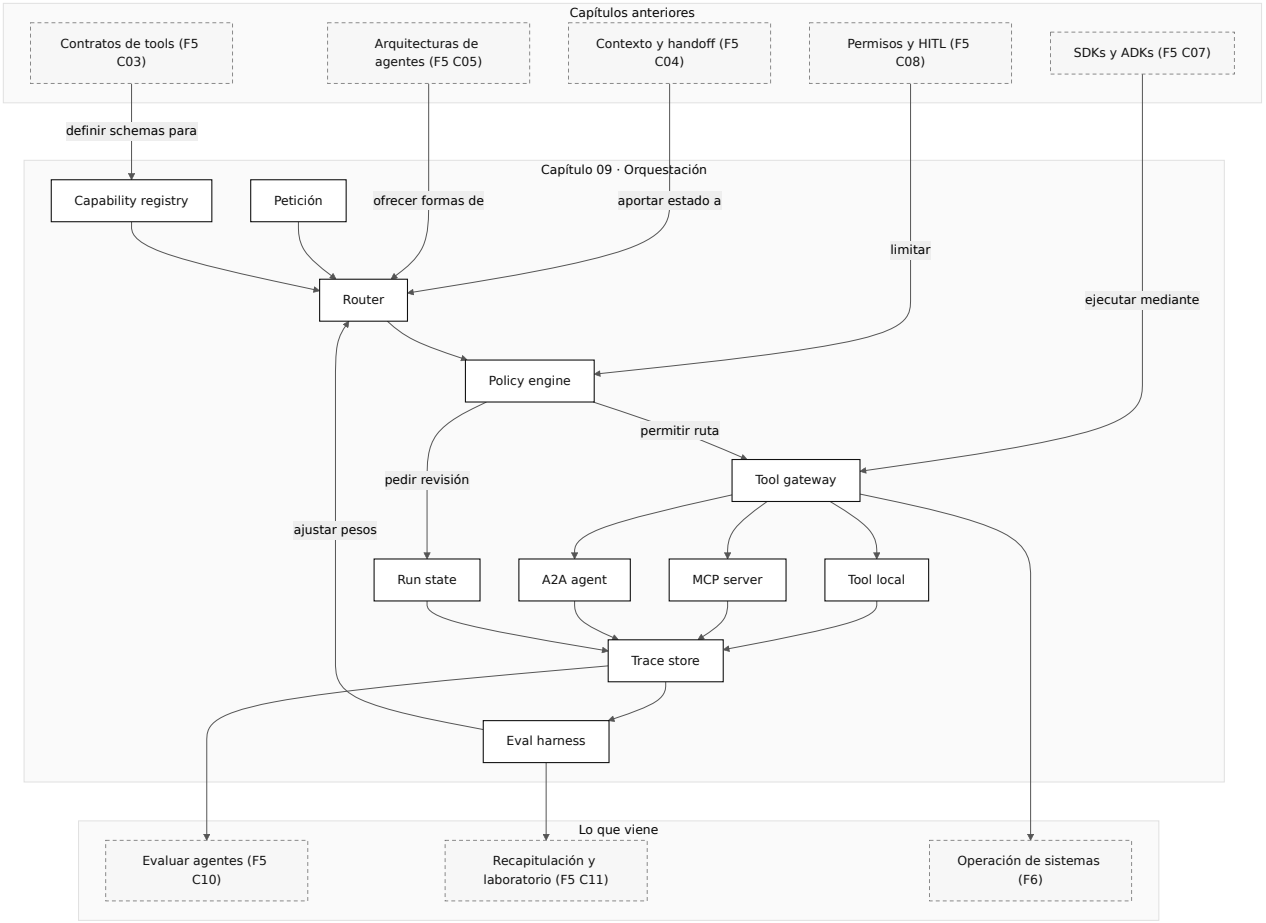
el hueco que deja esa confianza es un **contrato explícito**: qué tarea se puede pedir, en qué formato, con qué garantías de respuesta, qué pasa si el otro agente falla o tarda demasiado, y cómo se verifica que lo que devolvió es lo que se pidió.

Esto no es nuevo en informática; es la lección de décadas de sistemas distribuidos, ahora con un actor más impredecible al otro lado. Por eso protocolos como MCP, para exponer tools y contexto, o los esquemas de A2A, para coordinar agentes, no son «atajos» que te ahorran pensar: son justamente la formalización de esa frontera. Un servidor MCP es un contrato que dice «estas son las herramientas que ofrezco, con este esquema y estos permisos»; un intercambio A2A es un contrato que dice «esta tarea te la puedo encargar, y así sabremos los dos si salió bien». Cuanto más lejos vive la decisión de tu propio código, más explícito tiene que ser ese contrato, porque hay menos confianza implícita en la que apoyarse.

De aquí sale una regla de diseño que conviene llevarse a cualquier proyecto de orquestación. No trates una llamada a otro agente como si fuera una llamada a una función fiable: trátala como una integración con un servicio externo que puede fallar, mentir o tardar, y por tanto rodéala de las mismas defensas que rodearías a cualquier dependencia poco fiable. Un timeout, para que su lentitud no se vuelva tu lentitud. Una validación de lo que devuelve, porque su salida es entrada no confiable para ti, igual que una observación de tool. Un plan de recuperación, por si no responde. Y una traza que cruce la frontera, para que cuando algo falle entre los dos sistemas puedas saber de qué lado estuvo el problema. La orquestación madura no consiste en conectar agentes; consiste en diseñar las fronteras entre ellos como si cada una pudiera romperse, porque tarde o temprano alguna se rompe.



Cómo encaja todo



Vocabulario aprendido

TÉRMINO	DEFINICIÓN ÚTIL
Orquestación	Capa que decide ruta, agente, tool, permisos y estrategia de ejecución.
Router	Componente que selecciona una ruta usando señales observables.
Capability registry	Catálogo de capacidades con contrato, coste, latencia, versión y owner.
MCP	Protocolo para conectar hosts con herramientas, recursos y prompts.
MCP server	Servicio que expone capacidades mediante MCP.
Resource	Dato o contexto que un servidor MCP puede ofrecer.
Tool filtering	Exponer solo un subconjunto de tools a un agente.
Tool list	Lista de tools disponibles para un cliente o agente en un momento concreto.
Schema drift	Cambio de contrato que rompe supuestos de entrada o salida.
A2A	Protocolo para coordinar sistemas agentic independientes.
AgentCard	Manifiesto que describe identidad, capacidades, skills, interfaces y seguridad.
Task	Unidad de trabajo durable en A2A.
Artifact	Resultado producido por una tarea, separado de los mensajes.
Handoff	Transferencia de una tarea a otro agente, normalmente dentro de un runtime.
Fallback	Ruta alternativa cuando la primera falla antes de producir resultado útil.
Idempotencia	Propiedad que permite repetir una operación sin duplicar efectos.
RouteDecision	Recibo estructurado con ruta elegida, motivo, score y alternativas.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Llamar orquestación a cualquier cadena	Varias llamadas parecen arquitectura.	Exigir router, contrato, estado, permisos y traza.
Usar MCP para todo	Es cómodo exponer tools estándar.	Usar MCP cuando haya reutilización real y controlar tool filtering.
Usar A2A demasiado pronto	Suena moderno delegar a otro agente.	Usarlo solo si el destino tiene autonomía, estado y capacidades propias.
Dejar que el modelo elija siempre	Reduce código al principio.	Separar reglas de negocio, policy, routing y decisión del modelo.
No registrar alternativas	Solo guardas la ruta ganadora.	Guardar top candidatos y motivos de descarte.
Ignorar latencia de descubrimiento	Listar tools parece gratis.	Cachear tool list, versionar capacidades y medir p95.
Mezclar output conversacional y artefactos	Todo termina como texto.	Separar mensaje, task, artifact, diff, archivo y traza.
No ensayar fallos	La demo solo prueba el camino feliz.	Simular tool ausente, schema drift, timeout y task parcial.
Permitir fallback hacia más impacto	Parece una forma de “resolver como sea”.	Fallback solo hacia rutas de igual o menor efecto operativo.

Antes de pasar página

Antes del capítulo 10, deberías poder responder:

PREGUNTA	SI DUDAS, VUELVE A...
¿Qué diferencia hay entre orquestar y encadenar llamadas?	`Qué no es orquestar`.
¿Qué entra y qué sale de una función de orquestación?	`La definición útil`.
¿Por qué el routing debe registrar alternativas descartadas?	`Las tripas de la orquestación`.
¿Cómo se puntúa una ruta sin fingir precisión absoluta?	`Fórmula práctica para elegir ruta`.
¿Cuándo usarías regla, clasificador, LLM router o grafo?	`Routing: reglas, modelo o grafo`.
¿Qué problema resuelve MCP y qué no resuelve?	`MCP: tools y contexto como contrato externo`.
¿Qué cambia cuando usas A2A en lugar de una tool?	`A2A: cuando el destino también decide`.
¿Por qué un handoff interno no siempre es A2A?	`Handoffs, routing y A2A no son lo mismo`.
¿Qué árbol usarías para elegir entre tool local, MCP, handoff, A2A o workflow?	`Árbol de decisión para elegir arquitectura`.
¿Qué aspecto mínimo tienen una tool MCP, una AgentCard y una Task A2A?	`Contratos mínimos: qué viaja realmente`.
¿Qué fallos deberías ensayar antes de publicar?	`Fallos de producción que conviene ensayar`.
¿Qué decisiones de ingeniería hacen publicable la orquestación?	`Decisiones de ingeniería que no se ven en la demo`.
¿Cómo implementarías un router mínimo sin casarte con proveedor?	Practícalo en el cuaderno del facsímil.

Para saber más

- Agent2Agent Protocol. (2026). *Specification*. https://google-a2a.github.io/A2A/specification/
- Anthropic. (2026). *MCP Connector*. https://platform.claude.com/docs/en/agents-and-tools/mcp-connector
- Google. (2026). *ADK with Agent2Agent Protocol*. https://adk.dev/a2a/

- Google. (2026). *Agent Development Kit: MCP Tools*. https://adk.dev/tools-custom/mcp-tools/
- Google. (2026). *Agent Development Kit: Route Between Agents*. https://adk.dev/agents/routing/
- Google. (2026). *Agent Development Kit: Route Between Models*. https://adk.dev/agents/models/routing/
- Google. (2026). *Agent Development Kit: Template Agent Workflows*. https://adk.dev/agents/workflow-agents/
- Jennings, N. R., Sycara, K., & Wooldridge, M. (1998). *A roadmap of agent research and development*. https://doi.org/10.1023/A:1010090405266
- Keeney, R. L., & Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley.
- Model Context Protocol. (2026). *Specification*. https://modelcontextprotocol.io/specification
- Ong, I., Almahairi, A., Wu, V., Zhang, W., Lin, T., Zhang, R., Stoica, I., & Gonzalez, J. E. (2024). *RouteLLM: Learning to Route LLMs with Preference Data*. https://arxiv.org/abs/2406.18665
- OpenAI. (2026). *Agents SDK: Handoffs*. https://openai.github.io/openai-agents-python/handoffs/
- OpenAI. (2026). *Agents SDK: Model Context Protocol*. https://openai.github.io/openai-agents-js/guides/mcp/
- Smith, R. G. (1980). *The Contract Net Protocol*. https://doi.org/10.1109/TC.1980.1675516
- Wooldridge, M., & Jennings, N. R. (1995). *Intelligent agents: Theory and practice*. https://doi.org/10.1017/S0269888900008122

En resumen

IDEA	QUÉ TE LLEVAS
Orquestar es decidir rutas con contrato.	No basta con encadenar agentes: hay que registrar por qué se eligió cada ruta.
MCP y A2A resuelven problemas distintos.	MCP conecta tools y contexto; A2A coordina sistemas agentic independientes.
El router necesita señales, no intuición.	Encaje, calidad, disponibilidad, latencia, coste, riesgo y permisos deben verse en la decisión.
Los ADKs ayudan, pero no sustituyen arquitectura.	Puedes usar OpenAI, Claude, Google ADK o LangGraph, pero tu dominio debe mantener contratos propios.
La evaluación del capítulo 10 empieza aquí.	Sin trazas y alternativas descartadas no podremos medir si la orquestación funciona mejor.

Notas

1. Wooldridge, M., & 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. Jennings, N. R., Sycara, K., & 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. ↩

3. 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. ↩

4. Keeney, R. L. y Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley. Formaliza la función de valor aditiva ponderada para decidir entre alternativas. ↩

5. Ong, I., Almahairi, A., Wu, V., Zhang, W., Lin, T., Zhang, R., Stoica, I. y Gonzalez, J. E. (2024). *RouteLLM: Learning to Route LLMs with Preference Data*. https://arxiv.org/abs/2406.18665 Aprende a enrutar peticiones entre modelos según coste y calidad esperados. ↩

6. Google. (2026). *Agent Development Kit: Route Between Agents*. https://adk.dev/agents/routing/. Consultado el 10 de junio de 2026. ↩

7. Google. (2026). *Agent Development Kit: Route Between Models*. https://adk.dev/agents/models/routing/. Consultado el 10 de junio de 2026. ↩

8. Google. (2026). *Agent Development Kit: Template Agent Workflows*. https://adk.dev/agents/workflow-agents/. Consultado el 10 de junio de 2026. ↩

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

.0. OpenAI. (2026). *Agents SDK: Model Context Protocol*. https://openai.github.io/openai-agents-js/guides/mcp/. Consultado el 10 de junio de 2026. ↩

.1. Anthropic. (2026). *MCP Connector*. https://platform.claude.com/docs/en/agents-and-tools/mcp-connector. Consultado el 10 de junio de 2026. ↩

.2. Google. (2026). *Agent Development Kit: MCP Tools*. https://adk.dev/tools-custom/mcp-tools/. Consultado el 10 de junio de 2026. ↩

.3. Google. (2026). *ADK with Agent2Agent Protocol*. https://adk.dev/a2a/. Consultado el 10 de junio de 2026. ↩

.4. Agent2Agent Protocol. (2026). *Specification*. https://google-a2a.github.io/A2A/specification/. Consultado el 10 de junio de 2026. ↩

.5. OpenAI. (2026). *Agents SDK: Handoffs*. https://openai.github.io/openai-agents-python/handoffs/. Consultado el 10 de junio de 2026. ↩

.6. Anthropic. (2024). *Building Effective Agents*. https://www.anthropic.com/engineering/building-effective-agents Describe el patrón orquestador-trabajadores, donde un coordinador reparte el trabajo entre subagentes. ↩