LO ESENCIAL · **Módulo 10 · La organización de agentes**

Por qué «empresa con CEO» fracasa y cuál es la versión válida. Orquestador como máquina de estados, contratos JSON, humano en el punto irreversible.

10.1 · Por qué la metáfora «empresa con CEO» hace fracasar proyectos

Objetivos

- Hacer la autopsia de un diseño multiagente fallido
- Formular la versión válida de la metáfora

La metáfora que vende y fracasa

«Un agente CEO que delega en un agente de ventas, uno de contabilidad y uno de compras, y ellos se coordinan entre sí.» Suena bien en una diapositiva. En producción fracasa casi siempre, por razones que no son de moda sino de ingeniería:

Lo que la metáfora supone	Lo que pasa
Los roles «entienden» su cargo	entienden un prompt; con contexto insuficiente se salen del rol
Se coordinan hablando	el texto libre entre agentes pierde información y multiplica errores
El CEO decide bien	el CEO es un modelo sin estado, sin memoria de lo que ya se intentó
Más agentes = más capacidad	más agentes = más prefill, más latencia, más puntos de fallo
Se supervisan entre sí	dos modelos que se equivocan igual no se corrigen
Es «como una empresa»	una empresa tiene procedimientos escritos, estado en sistemas, y humanos que paran cosas

Autopsia de un diseño fallido

Caso (real, anonimizado): un «CEO» recibía «gestionar la factura», la pasaba a «contabilidad», que la pasaba a «compras» para validar el proveedor, que la devolvía al CEO con una pregunta, que la reenviaba… Diez pasos, 40 000 tokens de prefill, tres minutos, y **la factura se registró dos veces** porque nadie tenía el estado.

Con la versión válida: un **orquestador determinista** con cuatro estados (`recibida → extraída → validada → registrada`), un solo modelo llamado en dos de ellos con esquemas, y una regla de idempotencia por número de factura. Ocho segundos, un registro.

La versión válida de la metáfora

Lo que sí se parece a una empresa:

- **Procedimientos escritos** = máquina de estados (10.3).
- **Formularios entre departamentos** = contratos JSON (10.4).
- **Sistemas de registro** = estado persistido, no memoria del modelo.
- **Presupuestos** = topes duros por tarea.
- **Firmas** = el humano en el punto irreversible.

Y lo que **no**: un jefe que improvisa, empleados que se hablan sin formulario, y nadie que lleve el libro.

Entregable: `labs/10-01/autopsia.md`: tome un flujo multiagente publicado (o el propio) y escriba su autopsia con la tabla de arriba. Commit: «Capa 10.1: autopsia».

Nada; y el rediseño **mejora** más cuanto peor es el hardware, porque quita prefill.

Cuándo NO usar esto **Por qué la metáfora «empresa con CEO» hace fracasar proyectos**

- No use esta lección para decir «los agentes no sirven». Sirven cuando hay razones legítimas para dividir (lección siguiente) y un orquestador que no improvisa.

PRÁCTICA · **Práctica**

1. Cuente los prefills del caso fallido. ¿Cuánto costaría por día con 100 facturas en perfil B?
2. Reescriba el caso con un orquestador de cuatro estados.

Referencias

1. Anthropic (2024). *Building effective agents* — «flujos» contra «agentes».
2. Módulos 10.2–10.4 de este curso.

10.2 · Las cuatro razones legítimas para dividir en roles

Objetivos

- Justificar cada rol por permisos, contexto, modelo o paralelismo
- Rechazar roles que no cumplen ninguna

Solo hay cuatro razones para dividir en roles

Si un rol no se justifica con **al menos una** de estas, no existe: es prompt.

Razón	Qué resuelve	Ejemplo en la ferretería
1 · Permisos	un rol puede ver o hacer cosas que otro no debe	el rol que lee nómina no es el mismo que atiende al vendedor; el que **escribe** en la base es distinto del que **lee**
2 · Contexto	dos tareas necesitan contextos que no caben juntos o se estorban	extraer facturas (esquema + ejemplos de facturas) y redactar actas (plantilla + transcripción) no comparten ventana
3 · Modelo	tareas distintas rinden mejor con modelos distintos	clasificar el tipo de documento: modelo pequeño y rápido; extraer campos: modelo mediano; VLM solo para fotos
4 · Paralelismo	volumen: muchas unidades independientes a la vez	400 facturas de fin de mes: N auxiliares idénticos, cada uno con una factura

Los roles del asistente, justificados

Rol	Razón	Modelo	Permisos
Recepción	3 (pequeño y rápido)	clasificador	lectura de metadatos
Extractor de facturas	2 + 3	mediano, con esquema	lectura de PDF, escritura en `borradores`
Consultor con cita	1 + 2	mediano	lectura filtrada por grupos
Redactor de actas	2	mediano	lectura de transcripciones, escritura en `borradores`
Auxiliares de volumen	4	el mismo que el extractor, N instancias	igual que el extractor
Auditor	1 (independencia)	pequeño o mediano	solo lectura de trazas

Seis roles, cada uno con una razón escrita. Lo que **no** hay: un «CEO», un «gerente de proyecto», un «coordinador». Eso es el orquestador, y no es un modelo (10.3).

PROCEDIMIENTO · **Laboratorio 10.2**

`labs/10-02/roles.md`: la tabla de roles del asistente con razón, modelo, permisos, y para

> cada uno la frase «si se quita este rol, pasa X». Si no se puede escribir la frase, se quita el rol.
> **Entregable:** la tabla justificada. Commit: «Capa 10.2: roles con razón».

Qué se degrada en cada perfil

A — Razón 3 manda: casi todo con el modelo pequeño; razón 4 no aplica (no hay paralelismo)

B — Dos modelos cargados a la vez si caben; paralelismo bajo

C — Todos los roles con su modelo; paralelismo real

Cuándo NO usar esto **Las cuatro razones legítimas para dividir en roles**

- No cree un rol «para que quede ordenado». Orden es un directorio, no un agente.
- No dé al auditor permisos de escritura: deja de ser auditor.

PRÁCTICA · **Práctica**

1. Un cliente pide un rol «asistente del gerente». ¿Cuál de las cuatro razones cumple? ¿Cómo se descompone?
2. ¿Por qué el auditor debe ser independiente del modelo del extractor?

Referencias

1. Módulo 3.2 (grupos del SSO) y 6.3 (permisos en el índice): la razón 1 se apoya en ellos.
2. Módulo 4 — por qué cargar dos modelos a la vez depende del perfil.

10.3 · El orquestador como máquina de estados

> Objetivos
>
> - Escribir la máquina de estados del asistente
> - Persistir el estado y poner presupuestos duros

El orquestador no piensa: ejecuta

Un **orquestador** es la pieza que decide qué paso sigue. Si esa pieza es un modelo con un prompt («coordina a los agentes»), el sistema es tan fiable como la peor respuesta de ese modelo. Si es una **máquina de estados** escrita en YAML y ejecutada por código, es tan fiable como el código.

La definición

```
# maquina.yaml — el procedimiento escrito
tarea: procesar_factura
presupuesto: {pasos_max: 8, tokens_max: 20000, segundos_max: 120, cop_max: 0}
estados:
  recibida:     {accion: guardar_original, siguiente: clasificada}
  clasificada:  {rol: recepcion, salida: {tipo: enum, valores: [factura, remision, otro]},
                 siguiente: {factura: extraida, remision: extraida, otro: fallida}}
  extraida:     {rol: extractor, esquema: factura.schema.json, reintentos: 1,
                 siguiente: validada, si_falla: fallida}
  validada:     {accion: reglas_coherencia,
                 siguiente: {ok_alta_confianza: registrada, ok_baja_confianza: aprobacion_humana,
     error: fallida}}
  aprobacion_humana: {accion: encolar_humano, espera: true,
                 siguiente: {aprobada: registrada, rechazada: rechazada}}
  registrada:   {accion: escribir_base, idempotencia: numero, final: true}
  rechazada:    {final: true}
  fallida:      {accion: encolar_humano_con_motivo, final: true}
```

Tres propiedades que un «CEO» no tiene:

1. **Estado persistido.** Cada transición se escribe en la base **antes** de ejecutar la siguiente. Si el proceso muere, se retoma donde iba.
2. **Presupuesto duro.** Al superar cualquier tope, el estado pasa a `fallida` con motivo. Nunca «un paso más».
3. **Idempotencia.** `registrada` con el mismo `numero` no escribe dos veces.

> PROCEDIMIENTO · **Laboratorio 10.3 · el ejecutor**

```
# ejecutar.py — recorre la máquina; persiste cada transición; respeta el presupuesto
import yaml, json, time
M = yaml.safe_load(open("maquina.yaml"))
def ejecutar(tarea_id, entrada, db):
    estado = db.estado(tarea_id) or "recibida"; gastado = db.gastado(tarea_id)
    t0 = time.time()
    while not M["estados"][estado].get("final"):
        if gastado["pasos"] >= M["presupuesto"]["pasos_max"] or time.time()-t0 > M["
```

```
presupuesto"]["segundos_max"]:
        db.transicion(tarea_id, estado, "fallida", motivo="presupuesto"); return "
fallida"
    e = M["estados"][estado]
    if e.get("espera"): db.encolar(tarea_id, estado); return estado    # se retoma
cuando el humano actúe
    resultado, tokens = paso(e, entrada, db)                           # rol (modelo) o
 acción (código)
    gastado["pasos"] += 1; gastado["tokens"] += tokens
    siguiente = e["siguiente"] if isinstance(e["siguiente"], str) else e["siguiente"][
resultado]
    db.transicion(tarea_id, estado, siguiente, resultado=resultado, gastado=gastado)
    estado = siguiente
return estado
```

Corra 20 facturas; mate el proceso a la mitad; vuelva a correr. Lo que debe ver: **ninguna se procesa dos veces**, las pendientes se retoman en su estado, y las de baja confianza quedan en `aprobacion_humana`.
Entregable: `maquina.yaml`, `ejecutar.py`, y el registro de transiciones de las 20. Commit: «Capa 10.3: orquestador».

Qué se degrada en cada perfil

A — Igual; cada paso con modelo tarda más, y el presupuesto de segundos se ajusta
B — Igual

Cuándo NO usar esto **El orquestador como máquina de estados**

- No ponga un modelo a «decidir el siguiente estado» salvo dentro de un estado, con salida enum y esquema. La transición la hace el código.
- No omita el estado `fallida`: un sistema que no puede fallar limpiamente falla sucio.

PRÁCTICA · **Práctica**

1. Añada el estado `duplicada` (la factura ya existe) y sus transiciones.
2. ¿Qué pasa si el humano tarda una semana en aprobar? Diseñe el vencimiento.

Referencias

1. Harel, D. (1987). *Statecharts: A Visual Formalism for Complex Systems.* Science of Computer Programming.

2. Anthropic (2024). *Building effective agents* — «orchestrator-workers» como flujo, no como agente libre.

10.4 · Contratos JSON entre roles y el humano en el punto irreversible

> Objetivos
>
> - Definir contratos entre roles
> - Implementar la cola de aprobación humana

Nada cruza entre roles sin esquema

Lo que un rol le pasa a otro es un **contrato JSON** con esquema, no un párrafo. Tres razones: se valida, se registra, y cuando cambia el modelo de un rol el otro no se entera.

```
{ "$id": "contrato/→extractorvalidador",
  "type": "object", "additionalProperties": false,
  "required": ["tarea_id", "documento", "campos", "confianza", "origen"],
  "properties": {
    "tarea_id":  {"type": "string"},
    "documento": {"type": "string", "description": "id del original, nunca el contenido"},
    "campos":    {"$ref": "factura.schema.json"},
    "confianza": {"type": "object"},
    "origen":    {"type": "object", "properties": {"rol": {"type":"string"}, "modelo": {"type":"
     string"}, "version": {"type":"string"}}}
  }}
```

Regla: los contratos llevan **identificadores**, no contenidos. El validador no necesita el PDF; necesita saber cuál es.

El humano en el punto irreversible

Hay acciones que no se deshacen: registrar un pago, enviar un correo a un proveedor, borrar un documento. Esas **nunca** las ejecuta un rol solo. Van a una **cola de aprobación** donde una persona con permiso ve qué se va a hacer, con qué evidencia, y aprueba o rechaza. Todo lo

reversible (crear un borrador, mover a `revision/`) sí lo hace el sistema.

Acción	¿Reversible?	Quién
crear borrador de acta	sí	sistema
registrar factura en contabilidad	**no**	contadora aprueba
enviar correo al proveedor	**no**	gerente aprueba
mover PDF a `procesados/`	sí	sistema
borrar datos de un titular (2.3)	**no**	gerente aprueba + registro

> PROCEDIMIENTO · **Laboratorio 10.4 · la cola**
>
> Una tabla `aprobaciones` (tarea, acción, evidencia, quién puede, vence, decisión) y una página mínima detrás del proxy (3.2) donde la contadora ve las facturas de baja confianza **con el PDF al lado**, corrige un campo si hace falta, y aprueba. Al aprobar, la máquina (10.3) retoma.
> **Entregable:** `contratos/*.schema.json` para los 5 pares de roles, y la cola funcionando con los 3 usuarios. Commit: «Capa 10: agentes con contrato y humano».
> Nada. La cola es una tabla y una página.

Cuándo NO usar esto **Contratos JSON entre roles y el humano en el punto irreversible**

- No pida aprobación para todo: si el 90 % de las facturas pasan por la contadora, el sistema no ahorra nada. El umbral se ajusta con 7.4.
- No deje que el modelo «proponga» la decisión en la cola con texto persuasivo: muestra evidencia, no argumentos.

PRÁCTICA · **Práctica**

1. Escriba el contrato `recepción → extractor`.
2. La contadora está de vacaciones. ¿Qué pasa con la cola? Diseñe la delegación.

Referencias

1. JSON Schema — `$ref` y composición: https://json-schema.org/understanding-json-schema/
2. Módulo 11.2 — cada aprobación queda en la traza con quién y cuándo.