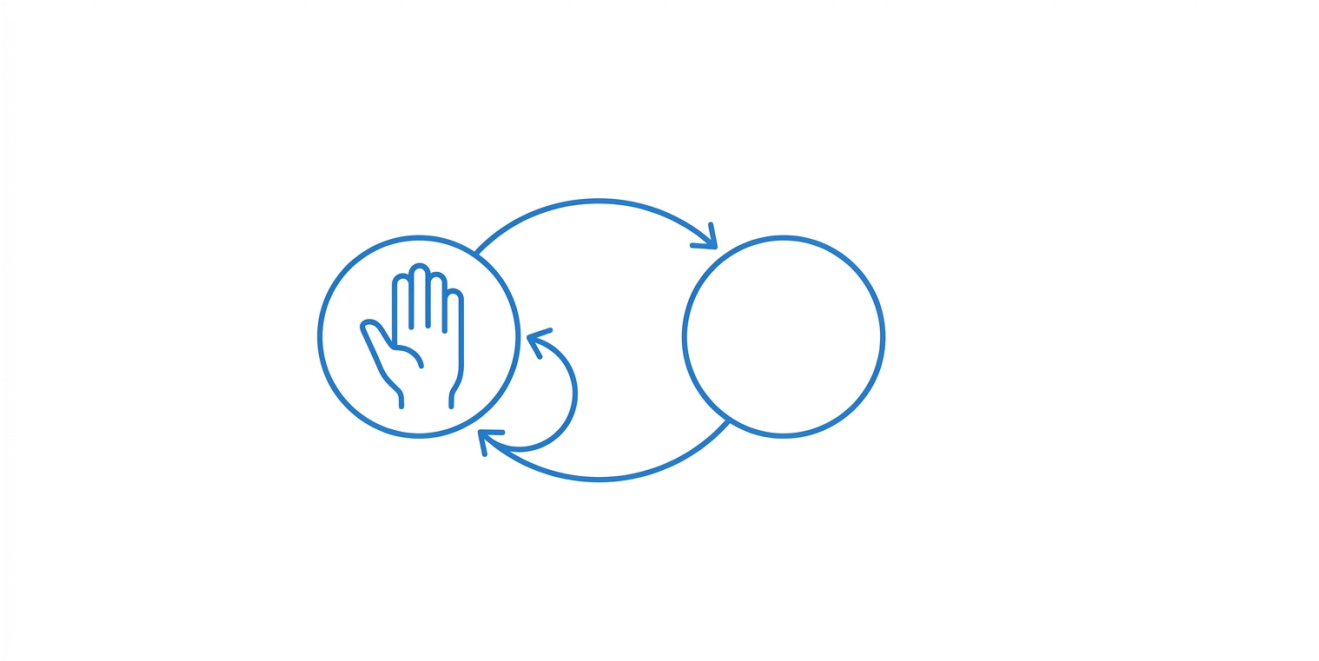
> **LO ESENCIAL · Módulo 10 · La organización de agentes**
>
> Por qué «empresa con CEO» fracasa y cuál es la versión válida. Orquestador como máquina de estados, contratos JSON, humano en el punto irreversible.

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



La organización de agentes

> 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 · Construir un agente paso a paso (y que la IA escriba los prompts)

El secretario del martes

Imagínese que el martes llega un secretario a la ferretería. Es rápido, no se cansa y no cobra por hora. Doña Marta le dice: «Mijo, revise los correos, y cada vez que llegue una factura de un proveedor, sáquele los datos y me la deja anotada. Si algo le parece raro, no invente: me pregunta».

Fíjese en todo lo que doña Marta le dio sin darse cuenta:

- Una **meta** («que las facturas queden anotadas»), no una orden suelta.
- Unas **herramientas**: el correo, el cuaderno donde anota, y usted para preguntarle.
- Un **límite**: si algo es raro, no decida solo.

Eso es un agente. Y la palabra que mejor lo describe es la que ya usa todo el mundo: un **secretario**. Un consultor responde una pregunta y se calla. Un secretario recibe un **encargo**, sale, hace diligencias y vuelve con el resultado. Y todo lo que vamos a construir en esta lección es exactamente ese secretario, pero con reloj —porque un secretario sin reloj puede pasarse el día releyendo el mismo correo.

Los cuatro objetos sobre la mesa

Antes de escribir nada, hay que tener claras cuatro cosas. Se escriben en media hoja, a mano.

El encargo. Una frase. «De cada factura de proveedor que llegue por correo, sacar número, fecha, proveedor, NIT y total, y dejarlos anotados. Lo dudoso, a Marta.» Si no cabe en una frase, son dos encargos.

Las herramientas. Tres o cuatro, no diez. Cada una con una etiqueta que diga exactamente qué hace *y cuándo no usarla*. «Buscar factura por número exacto — no sirve para buscar por fechas, para eso use la otra». La **caja de herramientas** del secretario: cada una etiquetada. Con veinte se confunde igual que uno humano.

La carta. Las casillas que la respuesta debe llenar. Cinco: número, fecha, proveedor, NIT, total. Ni una más. Y una casilla extra que casi nadie pone y lo cambia todo: **qué tan seguro está** de cada dato. Porque el secretario que dice «el total es 1.850.000, y de eso estoy seguro» y el que dice «creo que es 1.850.000, la tinta estaba corrida» merecen trato distinto.

El reloj. Cuántos pasos puede dar antes de rendirse y avisar. Ocho está bien. Sin esto, un agente se puede quedar dando vueltas toda la noche.

Los prompts los escribe la IA

Aquí viene lo que a todo el mundo le da pereza y no debería: **usted no escribe los prompts a mano**. Se los pide a un modelo, con una plantilla de tres líneas. Es más rápido y salen mejores que los suyos y los míos.

La plantilla es siempre la misma:

> Nota **Qué hace:** … **Qué recibe:** … **Qué devuelve:** …

Y el encargo que se le pasa al modelo es este —cópielo tal cual y cámbiele lo suyo:

```
Escríbame las instrucciones (system prompt) para un asistente que hace UNA sola tarea.

Qué hace: extraer los datos de una factura de proveedor colombiana.
Qué recibe: el texto de la factura, que puede venir de un escaneo y traer errores.
Qué devuelve: solo un JSON con estas casillas: numero, fecha, proveedor, nit, total,
              y confianza (0 a 1) por cada casilla.

Reglas que quiero que queden escritas en las instrucciones:
- Si un dato no está o no se lee, la confianza de ese dato es 0. NO lo invente.
- Los montos van en pesos, como número entero, sin puntos ni comas.
- No agregue casillas que no le pedí.
- No explique nada: solo el JSON.

Escríbalo en español, en segunda persona, en menos de 150 palabras.
```

Lo que devuelve, se pega y se prueba. Si falla, no se reescribe a mano: se le dice al modelo «falló en esto, corrija», y se vuelve a pegar. Ese ciclo dura minutos.

Paso a paso: el agente de doña Marta

1. Escriba el encargo en media hoja. Los cuatro objetos de arriba. No siga sin esto.

2. Pida los prompts. Uno por cada cosa que el agente tiene que *decidir*. En este caso dos: el que clasifica («¿esto es factura, remisión o basura?») y el que extrae. Use la plantilla.

3. Escriba la carta en JSON. Es el archivo más importante del agente. La palabra clave es `additionalProperties: false`, que en cristiano significa «no puede inventarse casillas».

```
{
  "type": "object",
  "additionalProperties": false,
  "required": ["numero","fecha","proveedor","nit","total","confianza"],
  "properties": {
    "numero":    {"type": "string"},
    "fecha":     {"type": "string", "format": "date"},
    "proveedor": {"type": "string", "minLength": 3},
    "nit":       {"type": "string"},
    "total":     {"type": "integer", "minimum": 0},
    "confianza": {"type": "object"}
  }
}
```

4. Escriba las herramientas. Cada una es una función normal de Python: buscar el correo, guardar en la base, avisar a Marta. Nada de IA aquí. Esto es plomería.

5. Arme el bucle, con reloj. Diez líneas. El agente ve el estado, decide, la herramienta corre, el resultado vuelve, y se repite. Si se acaba el reloj, se detiene y avisa.

6. Póngale el freno de mano. Todo lo que no se pueda deshacer —registrar un pago, mandarle un correo al proveedor— **no lo hace el agente**. Lo deja en una lista para que Marta apruebe. Lo reversible sí lo hace solo. Esta regla es la que permite dormir tranquilo.

7. Pruébelo con veinte casos de los que usted ya sabe la respuesta. Si acierta diecinueve, va. Si acierta doce, el problema casi nunca es el modelo: es la carta o el prompt. Vuelva al paso 2.

8. Déjelo trabajando y mírele el cuaderno. Cada cosa que hizo queda anotada: qué hizo, con qué documento, cuánto tardó, qué le costó. Ese cuaderno es lo que usted le muestra al cliente.

Lo que sale mal siempre

El agente que hace de todo. «Que también me conteste los correos y me arme el inventario». No. Un agente, un encargo. Dos encargos, dos agentes.

El agente sin reloj. Se lo van a encontrar a las seis de la mañana en el paso doscientos.

El agente con permiso de firmar. Nada irreversible sin una persona. Nunca.

El agente al que se le pide texto libre. «Dígame qué encontró» produce párrafos distintos cada vez y un programa que se cae. Casillas, siempre.

La trastienda

El bucle mínimo, con presupuesto duro:

```
def agente(modelo, herramientas, meta, max_pasos=8):
    ctx = [{"rol":"sistema","texto":INSTRUCCIONES}, {"rol":"usuario","texto":meta}]
    for _ in range(max_pasos):
        d = modelo(ctx, herramientas)              # → {"tipo":"texto"|"tool", ...}
        if d["tipo"] == "texto":
            return d["texto"]
        ctx.append({"rol":"tool","texto":str(ejecutar(d["tool"], d["args"]))})
    return "PRESUPUESTO AGOTADO — requiere revisión humana"
```

Presupuestos que se ponen en producción: `pasos_max` 6–10 · `tokens_max` ~20 000 por tarea · `segundos_max` 120. Al superarlos: estado `fallida` con motivo, nunca «un paso más».

Contrato entre roles: JSON con `tarea_id`, identificadores (no contenidos), `campos`, `confianza` y `origen{rol, modelo, version}`. Umbral de confianza inicial 0,8 → por debajo, cola humana.

Ver módulos 5.2 (esquemas), 10.3 (máquina de estados), 10.4 (contratos y aprobación), 12 (evals).

10.3 · Orquestar, mantener y actualizar sin romper nada

El procedimiento pegado en la pared

En la ferretería hay un papel pegado detrás del mostrador, amarillento, que dice qué hacer cuando llega mercancía: recibir, contar contra la remisión, si falta algo llamar al proveedor, si está completo firmar, guardar la remisión en la carpeta azul. Lo escribió el papá del gerente hace quince años. Nadie lo discute. Cuando entra un empleado nuevo, se le señala el papel.

Ese papel es un orquestador.

Cuando la gente monta sistemas con varios agentes, casi siempre hace lo contrario: pone un **agente jefe** que «coordina» a los demás. Suena bien y fracasa, por una razón sencilla: el jefe también es un empleado que no recuerda nada. No sabe qué se intentó ya, no lleva la cuenta, y si un día contesta distinto, todo el proceso sale distinto. Un caso real: una factura registrada dos veces porque ningún agente sabía que ya se había registrado.

La versión que sí funciona no es un jefe. Es el papel de la pared: **estados escritos**. La factura está *recibida*, o *extraída*, o *validada*, o *esperando a Marta*, o *registrada*. De cada estado sale una flecha a otro, y esas flechas las decide el código, no el modelo. El modelo solo trabaja **dentro** de un estado —extrae, clasifica—, y lo que dice se anota antes de pasar al siguiente.

Eso tiene tres consecuencias que valen más que cualquier modelo grande:

Si se va la luz a la mitad, al volver el sistema mira en qué estado quedó cada factura y sigue desde ahí. **No repite lo hecho.**

Si la misma factura llega dos veces, el estado *registrada* ya existe para ese número, y no se registra de nuevo.

Y si algo falla, uno mira el papel y sabe **en cuál de los cinco pasos** buscar.

Mantener: el pajarito de la mina

Los mineros bajaban con un canario. Si el pájaro se quedaba callado, salían. No porque el pájaro supiera de gases: porque era pequeño y sensible, y avisaba antes que las personas.

Un sistema con IA se rompe de una forma particularmente traicionera: **sigue funcionando**. El proveedor cambia el formato de la factura y el extractor empieza a sacar el total equivocado, sin error, sin alarma, con toda seguridad. Un mes después alguien cuadra la caja y no cuadra.

Por eso todas las noches, a las cuatro de la mañana, corre un pajarito: quince preguntas de las que ya se sabe la respuesta. La factura de siempre, que siempre da 1.850.000. La consulta de siempre, que siempre debe traer la cláusula siete. Y una prueba de que el vendedor **no** puede ver la nómina. Si alguna cambia respecto a anoche, le llega un mensaje al celular a las 4:15.

Un canario que escribe en un archivo que nadie lee es un pájaro muerto. Tiene que avisar por donde usted sí mira.

Actualizar sin romper

Llega un modelo nuevo. Todo el mundo dice que es mejor. ¿Se cambia?

En la ferretería nadie cambia el proveedor de tornillos porque un vendedor diga que los suyos son mejores. Se pide una caja, se prueban, y se compara con los de siempre. Aquí igual, y el procedimiento es de tres pasos:

Antes. Se corre el examen completo —los sesenta casos— con lo que hay hoy. Se guarda la nota.

Un solo cambio. Se cambia **una** cosa: el modelo, o el prompt, o el buscador. Nunca dos. Si se cambian dos y la nota sube, usted no sabe cuál sirvió; y si baja, tampoco.

Después. Se corre el mismo examen. Y la regla que decide, sin discusión: entra si sube en lo que se quería mejorar **y no baja en nada más**. Si la extracción de facturas sube tres puntos pero las actas bajan cinco, no entra —o entra solo para facturas.

Lo mismo aplica a las actualizaciones de sistema, a los drivers, a todo. Y si algo se actualizó solo y nadie se dio cuenta, el pajarito avisa esa misma noche.

Paso a paso: montar el procedimiento

1. **Dibuje los estados en una hoja.** Círculos y flechas. Cinco o seis, no veinte. Si necesita veinte, son dos procesos.
2. **Marque en qué estados trabaja el modelo.** Suelen ser uno o dos. Los demás son código.
3. **Escríbalo en un archivo de texto** (YAML), con el presupuesto arriba: pasos, tiempo, tokens.
4. **Guarde el estado en la base**, no en la memoria del programa. Antes de pasar al siguiente.
5. **Ponga la idempotencia**: registrar dos veces el mismo número de factura no debe poder pasar.
6. **Escriba el canario**: tres pruebas con respuesta conocida. Prográmelo de madrugada.
7. **Provoque una falla a propósito** —cambie el formato de la factura de prueba— y verifique que la alerta llega. Un canario sin probar es un pájaro decorativo.
8. **Escriba el examen de sesenta casos** y córralo. Esa nota es su línea base para siempre.

La trastienda

```
tarea: procesar_factura
presupuesto: {pasos_max: 8, tokens_max: 20000, segundos_max: 120}
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: {alta: registrada, baja: aprobacion_humana, error: fallida}}
  aprobacion_humana: {accion: encolar_humano, espera: true,
                siguiente: {aprobada: registrada, rechazada: rechazada}}
  registrada:  {accion: escribir_base, idempotencia: numero, final: true}
```

Canario nocturno: 15 casos (permisos, trampa, 5 facturas, 5 consultas), compara con la corrida anterior, alerta si baja o si falla cualquier caso de permisos. Cron a las 4:15 con `timeout`.

Actualizar: `antes.json` → un cambio → `despues.json` → entra solo si sube el objetivo y no baja ninguna otra métrica. Ver módulos 12.1–12.4.

10.4 · 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.

10.5 · 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.

10.6 · 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.