REVISTA TÓPICOS

https://revistatopicos.com.br – ISSN: 2965-6672

UM ORQUESTRADOR DETERMINÍSTICO POR MÁQUINA DE ESTADOS PARA WORKFLOWS MULTIAGENTES COM COMUNICAÇÃO BASEADA EM FILAS

DOI: 10.5281/zenodo.18716524

André de Mattos Ferraz

RESUMO

Fluxos multiagentes conectados por filas de mensagens são cada vez mais adotados em ambientes corporativos por sua modularidade e escalabilidade e, mais recentemente, tornaram-se um padrão comum em sistemas de inteligência artificial generativa (GenAI), nos quais modelos de linguagem de grande porte (LLMs) atuam como agentes que utilizam ferramentas e realizam etapas de roteamento e decisão. Entretanto, concorrência, falhas parciais e o comportamento probabilístico dos modelos introduzem não determinismo, prejudicando reprodutibilidade, auditabilidade e confiabilidade operacional. Propomos um orquestrador determinístico baseado em uma máquina de estados explícita, no qual os agentes são restritos a processamento puro: eles não decidem o próximo estágio nem transportam payloads grandes nas mensagens. Em vez disso, o orquestrador avalia condições de transição, impõe idempotência e registra uma trilha de eventos auditável, roteando artefatos por referências. Definimos métricas de

determinismo operacional — estabilidade do caminho de estados, taxa de sucesso de replay e escore de divergência — e comparamos a abordagem proposta a baselines de coreografia e de “agente decide o próximo estágio” sob múltiplas cargas (fluxos lineares, ramificação condicional, falhas injetadas e alta concorrência). Os resultados mostram que a orquestração determinística reduz substancialmente execuções divergentes e processamento órfão, melhora a capacidade de replay e reduz o esforço de recuperação, com sobrecarga limitada em latência e armazenamento de eventos. Esses achados sugerem que a reprodutibilidade em sistemas multiagentes com LLMs é alcançada não pela eliminação de componentes probabilísticos, mas pela restrição do controle de fluxo e pela auditabilidade por construção.

Palavras-chave: workflows multiagentes. inteligência artificial generativa (GenAI). LLMs. orquestração. máquinas de estados. filas de mensagens. auditabilidade. determinismo. idempotência.

ABSTRACT

Multi-agent pipelines connected through message queues are increasingly adopted in enterprise environments due to modularity and scalability, and more recently have become a common pattern for generative AI systems where large language models (LLMs) act as tool-using agents that make routing and decision steps. However, concurrency, partial failures, and probabilistic model behavior introduce non-determinism that harms reproducibility, auditability, and operational reliability. We propose a deterministic orchestrator based on an explicit state machine where agents are restricted to pure processing: they do not decide the next stage nor carry

large payloads within messages. Instead, the orchestrator evaluates transition conditions, enforces idempotency, and records an auditable event trail while routing artifacts by references. We define operational determinism metrics—state-path stability, replay success rate, and divergence score—and compare the proposed approach against choreography-based and agent-decides-next baselines under multiple workloads (linear flows, conditional branching, injected faults, and high concurrency). Results show that deterministic orchestration substantially reduces divergent executions and orphan processing, improves replayability, and lowers recovery effort with limited overhead in latency and event storage. These findings suggest that reproducibility in multi-agent LLM systems is achieved not by eliminating probabilistic components, but by constraining control flow and making execution auditable by design.

Keywords: multi-agent workflows. generative AI (GenAI). LLMs. orchestration. state machines. message queues. auditability. determinism. idempotency.

1. INTRODUÇÃO

Workflows multiagentes baseados em filas de mensagens são amplamente utilizados para implementar automação corporativa complexa. Decompor um pipeline em agentes independentes permite desenvolvimento paralelo, isolamento de falhas e escalabilidade flexível. Na prática, porém, muitas equipes observam que implantações multiagentes entram em uma espécie de “purgatório do piloto”: execuções repetidas com as mesmas entradas produzem resultados diferentes, o debug se torna demorado e requisitos de

conformidade (rastreabilidade, justificativa, evidências) são difíceis de atender.

Essa instabilidade é frequentemente atribuída ao caráter probabilístico dos modelos de IA, mas a causa raiz é mais ampla. Pipelines reais combinam mensageria assíncrona, retentativas, concorrência, consistência eventual, falhas parciais, timeouts e etapas heterogêneas (por exemplo, OCR, validação, enriquecimento). Quando o controle de fluxo é descentralizado — seja por coreografia direta (cada agente encaminha para o próximo) ou por permitir que agentes decidam o próximo estágio — pequenas diferenças de tempo e particularidades de tratamento de erros podem alterar o caminho de execução. Isso leva a trajetórias de estado não determinísticas, processamento duplicado, mensagens órfãs e artefatos irreproduzíveis.

Argumentamos que a principal alavanca para melhorar a confiabilidade é o controle de fluxo determinístico: um único componente deve possuir a semântica do workflow, impor idempotência e persistir uma trilha de auditoria que viabilize replay e análises pós-mortem. Propomos, portanto, um Orquestrador Determinístico por Máquina de Estados (DSMO) para workflows multiagentes baseados em filas. O DSMO centraliza decisões de roteamento e condições de transição em uma máquina de estados versionada, enquanto os agentes permanecem como processadores sem estado. As mensagens carregam apenas referências a artefatos e metadados de execução; o orquestrador registra eventos por estágio para auditabilidade.

Contribuições.

1. **Arquitetura:** DSMO, um orquestrador determinístico por máquina de estados para workflows multiagentes baseados em filas, separando controle de fluxo da lógica dos agentes.

2. **Modelo de mensagens e eventos:** um esquema mínimo, porém auditável, com referências a artefatos, identificadores de correlação, versionamento e chaves de idempotência.

3. **Métricas de determinismo:** medidas operacionais (estabilidade do caminho de estados, taxa de sucesso de replay e escore de divergência) que quantificam reprodutibilidade além do output final.

4. **Avaliação:** experimentos comparativos contra baselines de coreografia e de "agente decide o próximo estágio" em múltiplas cargas (ramificação, falhas injetadas e alta concorrência), mostrando maior reprodutibilidade e confiabilidade operacional com sobrecarga moderada.

2. DESENVOLVIMENTO

2.1. Definição do Problema e Objetivos

Modelamos um workflow como um grafo direcionado G = (S, T), onde S é um conjunto finito de estágios (estados) e T é um conjunto de transições. Cada transição $t \in T$ é definida por $(s_i \rightarrow s_j, \varphi)$, onde φ é uma condição booleana avaliada sobre metadados da execução e outputs de estágios anteriores.

Uma execução do pipeline (run) é disparada por um artefato de entrada a_0 e produz uma sequência de execuções de estágios $\pi = [sk_1, sk_2, \ldots, sk_n]$. Em ambientes assíncronos, concorrência e retentativas podem fazer múltiplas execuções candidatas competirem ou serem reexecutadas.

Determinismo Operacional.

Definimos determinismo operacional como a propriedade de que, para um artefato de entrada fixo e versões fixas dos componentes (definição do workflow, versões dos agentes e configuração), o orquestrador produz uma trajetória de estágios estável e artefatos intermediários estáveis dentro de limites aceitáveis de variância.

Objetivos de Projeto.

- **G1 — Controle de fluxo determinístico:** o roteamento entre estágios é decidido por uma máquina de estados versionada, não por agentes.
- **G2 — Auditabilidade:** todas as transições e resultados de estágios são registrados como eventos imutáveis.
- **G3 — Reexecução (replay):** deve ser possível reexecutar uma run (ou subconjuntos de estágios) de forma determinística.
- **G4 — Robustez operacional:** suportar retentativas, deduplicação, DLQ e tratamento de falhas parciais.
- **G5 — Segurança de mensagens:** evitar payloads grandes em mensagens; usar referências a artefatos.

2.2. Abordagem Proposta: DSMO

2.2.1. Visão Geral de Arquitetura

O DSMO é composto por:

- **Orquestrador:** avalia transições, enfileira próximos estágios, impõe idempotência e registra eventos.
- **Agentes (workers):** consomem mensagens de estágio, processam referências de artefatos de entrada, gravam artefatos de saída e emitem eventos de conclusão.
- **Fila(s):** uma por estágio (ou uma fila compartilhada com roteamento por estágio).
- **Repositório de artefatos:** armazenamento durável para entradas/saídas (ex.: object storage, banco de dados).
- **Log de eventos:** armazenamento append-only de eventos de run e estágio para auditoria e replay.

Restrição-chave: agentes nunca selecionam o próximo estágio. Sua responsabilidade termina ao produzir um artefato de saída e reportar o status.

2.2.2. Contrato de Mensagem (payload da Fila)

Um esquema mínimo de mensagem:

```
run_id: identificador único da execução
correlation_id: id de rastreamento ponta-a-ponta
stage: nome do estágio atual
workflow_version: hash/semver da máquina de estados
agent_version: identificador do build do agente
artifact_in_uri: ponteiro para o artefato de entrada
artifact_out_uri: ponteiro (placeholder) ou local es
idempotency_key: chave determinística, ex.: hash(run
attempt: número da tentativa
created_at: timestamp
config_ref: ponteiro para snapshot de configuração
```

Observação: nenhum texto completo/payload grande é carregado na mensagem.

2.2.3. Modelo de Eventos (trilha de Auditoria)

Eventos são persistidos como registros imutáveis:

- RUN_CREATED
- STAGE_ENQUEUED
- STAGE_STARTED
- STAGE_SUCCEEDED
- STAGE_FAILED

- RUN_COMPLETED
- RUN_ABORTED

Cada evento inclui run_id, stage, timestamps, URIs de artefatos, attempt e versões.

A trilha segue o padrão de event sourcing, registrando mudanças de estado como uma sequência append-only de eventos (FOWLER, 2005).

2.2.4. Idempotência e Deduplicação

Entrega “at-most-once” é inviável na maioria das filas; o DSMO implementa semântica efetivamente-uma-vez por:

- usar idempotency_key para impedir que o mesmo resultado de estágio seja aplicado múltiplas vezes,
- gravar outputs em caminhos determinísticos (ou fazer upsert),
- registrar STAGE_SUCCEEDED com restrição única (run_id, stage) (ou (run_id, stage, logical_partition)).

Isso se alinha ao padrão Idempotent Receiver (HOHPE; WOOLF, 2004).

2.2.5. Avaliação de Transições

Transições são avaliadas pelo orquestrador usando condições explícitas φ. As condições devem depender apenas de:

- status de estágios anteriores,
- metadados da run,
- outputs resumidos como metadados estruturados pequenos (não o payload completo), garantindo roteamento determinístico.

2.3. Algoritmo do Orquestrador (pseudocódigo)

```
AoReceberEvento(EstagioConcluido e):
  assert e.workflow_version == current_workflow_vers

  if IsDuplicateCompletion(e.run_id, e.stage, e.ider
      RecordEvent(DUPLICATE_IGNORED)
      return

  PersistEvent(e) // STAGE_SUCCEEDED ou STAGE_FAILEI

  if e.status == FAILED:
        next = FailurePolicy(e.run_id, e.stage, e.at
        if next == RETRY:
        EnqueueStage(e.run_id, e.stage, attempt=e.at
        RecordEvent(STAGE_ENQUEUED)
        else:
        SendToDLQ(e)
        RecordEvent(RUN_ABORTED)
        return
```

```
// sucesso
s_next = EvaluateTransitions(e.run_id, e.stage, work
if s_next == TERMINAL:
        RecordEvent(RUN_COMPLETED)
        return

EnqueueStage(e.run_id, s_next, attempt=1)
RecordEvent(STAGE_ENQUEUED)
```

2.4. Método

2.4.1. Visão Geral

Estudamos orquestração determinística para workflows multiagentes baseados em filas. Nossa abordagem, DSMO, centraliza a semântica do workflow em uma máquina de estados versionada e restringe agentes a processamento puro. Os agentes trocam apenas referências a artefatos via mensagens; o orquestrador impõe idempotência, aplica políticas de retentativa/DLQ e registra uma trilha imutável de eventos que habilita replay e auditoria.

Situamos o DSMO no contexto mais amplo de abordagens de orquestração com engines de workflow (ex.: NADEEM; MALIK, 2022).

Comparamos o DSMO contra dois baselines:

- **B1 (Coreografia):** cada agente roteia enfileirando o próximo estágio diretamente após concluir seu trabalho.
- **B2 (Agente decide o próximo):** agentes podem sobrescrever a decisão de roteamento, simulando heurísticas locais ou decisões ambíguas.

Todos os métodos executam na mesma infraestrutura local: uma fila durável, um repositório de artefatos e um log de eventos compartilhado.

2.5. Desenho Experimental

2.5.1. Baselines

- **B1 — Coreografia:** cada agente encaminha diretamente para o próximo estágio.
- **B2 — Agente decide o próximo:** o agente seleciona next_stage com base em sua lógica interna.
- **B3 — DSMO (nossa proposta):** orquestrador avalia transições centralmente + eventos auditáveis + idempotência.

2.5.2. Cargas de Trabalho

- **W1 — Linear:** A → B → C
- **W2 — Condicional:** A → (B ou C) → D
- **W3 — Injeção de falhas:** timeouts aleatórios e falhas transitórias em B e C

- **W4 — Alta concorrência:** N runs em paralelo com backpressure e retentativas

2.5.3. Métricas

Determinismo.

- **Estabilidade do caminho de estados (SPS):** fração de replays que seguem a mesma sequência de estados.
- **Taxa de sucesso de replay (RSR):** fração de replays que atingem sucesso terminal.
- **Escore de divergência (DS):** 0/1 para mismatch de estágio terminal, com distância opcional para outputs estruturados.

Confiabilidade operacional.

- **Taxa de órfãos:** fração de execuções de estágio não vinculadas a uma trajetória válida.
- **Taxa de processamento duplicado:** fração de conclusões duplicadas de estágio.
- **Taxa de DLQ:** mensagens enviadas para DLQ a cada 1000 runs.

Sobrecarga.

- **Latência ponta-a-ponta:** p50/p95

- **Eventos por run e bytes armazenados por run**
- **Vazão:** runs/min

2.6. Resultados

2.6.1. Determinismo e Replayabilidade

Conforme mostrado na Tabela 1, o DSMO atinge alta replayabilidade mantendo trajetórias de execução estáveis. Em particular, o DSMO alcança SPS de 0,7775, levemente acima da coreografia (0,775) e substancialmente acima de agente-decide (0,56). De forma similar, o DSMO obtém RSR de 0,8725, superando coreografia (0,8525) e agente-decide (0,855). Embora a taxa base de sucesso ponta-a-ponta seja comparável entre DSMO (0,88) e coreografia (0,89), o DSMO fornece garantias mais fortes sobre o caminho percorrido e a capacidade de reproduzi-lo sob concorrência e retentativas; em contraste, agente-decide apresenta SPS marcadamente menor, indicando divergência frequente nas trajetórias de estado realizadas.

Tabela 1. Resumo de métricas de determinismo, confiabilidade e latência por modo de orquestração.

Métrica	DSMO	Coreografia	Agente-decide
Taxa base de sucesso	0.88	0.89	0.83
SPS	0.7775	0.775	0.56

RSR	0.8725	0.8525	0.855
Duplicatas ignoradas / run	0.165	0.19	0.22
Eventos órfãos / run	0.0	3.735	0.96
DLQ / run	0.12	0.11	0.17
Latência p50 (s)	0.241	0.205	18.626
Latência p95 (s)	0.540	0.537	19.296
Latência média (s)	0.273	0.252	17.941

2.6.2. Confiabilidade Operacional

O DSMO elimina processamento órfão nas cargas avaliadas, com 0,0 eventos órfãos por run, contra 3,735 na coreografia e 0,96 em agente-decide. Esse resultado é consistente com a escolha de centralizar decisões de transição e registrar eventos por estágio, permitindo atribuir cada execução a uma trajetória válida.

Em todos os modos, conclusões duplicadas são observadas (e precisam ser tratadas) devido a retentativas e entregas concorrentes. O DSMO registra 0,165 conclusões duplicadas ignoradas por run, menor que coreografia (0,19) e agente-decide (0,22), sugerindo que chaves explícitas de idempotência combinadas com roteamento centralizado reduzem trabalho redundante mesmo quando não há entrega at-most-once.

A pressão de DLQ permanece dependente da carga e é comparável entre DSMO (0,12 eventos DLQ por run) e coreografia (0,11), enquanto agente-decide apresenta a maior taxa (0,17). Esses resultados indicam que controle de fluxo orientado a determinismo não aumenta materialmente o uso de DLQ e pode reduzi-lo em relação ao roteamento conduzido por agentes.

2.6.3. Sobrecarga

O DSMO introduz sobrecarga modesta de latência em relação à coreografia: a mediana aumenta de 0,205 s (coreografia) para 0,241 s (DSMO), enquanto a cauda permanece similar (p95 0,537 s vs. 0,540 s). A média segue o mesmo padrão (0,252 s vs. 0,273 s). Em contraste, agente-decide incorre em latências ordens de grandeza maiores (p50 18,626 s, p95 19,296 s, média 17,941 s), consistente com trabalho adicional de controle de fluxo embutido nos agentes e maior incidência de divergência e recuperação.

No geral, o DSMO entrega melhor determinismo e confiabilidade operacional (notadamente, eliminando eventos órfãos) a um pequeno custo na latência mediana e sem aumento substancial na latência de cauda, superando agente-decide em todas as métricas reportadas.

2.6.4. Ameaças à Validade, Limitações e Recomendações de Uso

Recomendações práticas. Para adoção incremental, recomenda-se iniciar com: (1) centralizar o roteamento no orquestrador (mesmo que ainda sem métricas avançadas); (2) introduzir idempotency_key e unicidade por estágio; (3) registrar eventos mínimos para auditoria; e (4) migrar decisões de agentes para outputs estruturados validados. Em pipelines com LLMs,

recomenda-se também fixar parâmetros de inferência quando possível, registrar a versão do prompt e, se a plataforma suportar, usar seeds ou mecanismos equivalentes para reduzir variância do plano de dados. Mesmo quando a saída final continuar probabilística, o workflow torna-se mais previsível, depurável e auditável.

Quando usar DSMO vs. coreografia. Em ambientes pequenos, de baixa criticidade e baixa concorrência, coreografia pode ser suficiente e mais simples. O DSMO se justifica quando há necessidade de: (i) rastreabilidade e auditoria formal; (ii) replay confiável para reprocessamento; (iii) mitigação de duplicatas e mensagens órfãs; (iv) evolução versionada do workflow; e (v) convivência com probabilismo (LLMs) sem transformar isso em caos operacional. Em outras palavras, o DSMO é mais valioso quando o custo de "execuções divergentes" é alto — seja por compliance, seja por custo computacional, seja por tempo de recuperação.

Generalização dos resultados experimentais. Os workloads avaliados (linear, condicional, falhas injetadas e alta concorrência) são representativos de muitos pipelines, mas não cobrem todas as classes de sistemas corporativos. Workflows com dependências humanas (human-in-the-loop), janelas de tempo rígidas, ou integração com transações financeiras podem exigir políticas mais conservadoras (por exemplo, travas por run, checkpoints explícitos e aprovação manual antes de certos estágios). Uma extensão natural é avaliar o DSMO em cenários de produção com volumes maiores e heterogeneidade de agentes (modelos diferentes, linguagens diferentes, vendors diferentes).

Condições de corrida fora do orquestrador. O DSMO reduz corridas no roteamento, mas não elimina corridas no mundo externo (por exemplo, chamadas a APIs, gravações concorrentes no storage ou dependências com consistência eventual). Para manter determinismo operacional, é recomendável que efeitos externos sejam encapsulados por: (i) idempotency keys também no lado do provedor, quando possível; (ii) padrões de outbox ou saga para efeitos distribuídos; e (iii) escrita de outputs em paths determinísticos com compare-and-set quando suportado. Quando essas garantias não existem, o replay pode repetir efeitos colaterais indesejados — e isso deve ser considerado no desenho do workflow.

Custo de audit trail e retenção. Uma trilha de eventos append-only aumenta armazenamento e exige política de retenção. O custo, porém, pode ser controlado: eventos são pequenos e podem ser compactados/particionados por período. Além disso, o ganho em auditabilidade e capacidade de replay frequentemente compensa o overhead, especialmente em ambientes regulados. Na prática, recomenda-se separar “eventos essenciais” (RUN_CREATED, STAGE_ENQUEUED, STAGE_* e RUN_COMPLETED/ABORTED) de “eventos estendidos” (metadados de inferência, estatísticas, checks), com níveis configuráveis.

Dependência de bons contratos e outputs estruturados. O DSMO pressupõe que agentes produzam outputs estruturados mínimos para avaliação de condições de transição. Em ambientes de GenAI, isso significa que um agente LLM deve retornar JSON válido e validável (por schema) sempre que sua saída impactar decisões. Se a saída do agente for livre e ambígua, a condição φ pode se tornar frágil e introduzir “não determinismo por

parsing". Assim, recomenda-se: validação rígida, fallback determinístico (por exemplo, rota FAIL_SAFE) e registro explícito de violações de contrato.

Embora os resultados experimentais indiquem ganhos em reprodutibilidade e confiabilidade, há limitações e ameaças à validade que precisam ser explicitadas. Em primeiro lugar, as métricas SPS/RSR/DS são operacionais e dependem do desenho do workflow e do que é considerado "divergência aceitável". Em pipelines com etapas probabilísticas (como LLMs), a estabilidade do artefato final pode ser baixa mesmo quando a trajetória de estados é estável. Portanto, é importante separar: (i) determinismo do plano de controle (trajetória de estados, dedup, idempotência, replay) e (ii) variância do plano de dados (conteúdo gerado). O DSMO mira principalmente o primeiro.

2.7. Implementação de Referência e Considerações Operacionais

Determinismo em pipelines com LLMs. Em sistemas multiagentes com LLMs, parte do não determinismo vem do modelo e parte vem do plano de controle. O DSMO não "conserta" a aleatoriedade intrínseca do modelo, mas impede que ela se transforme em divergência estrutural do workflow. Ao bloquear agentes de decidir o próximo estágio e ao forçar transições baseadas em condições determinísticas (sobre estados e metadados estruturados), o DSMO reduz a variância operacional: o que muda tende a ser o conteúdo do artefato, não o caminho executado. Isso é particularmente relevante quando LLMs são usadas como "roteadores" (routing), porque pequenas variações textuais podem mudar decisões locais; no DSMO, essa

decisão pode ser convertida em um output estruturado e validado, e então consumida pelo orquestrador sob regras claras.

Observabilidade e rastreabilidade ponta-a-ponta. O DSMO facilita rastreio ao exigir correlation_id e run_id em todas as mensagens e eventos. Métricas mínimas incluem: taxa de duplicatas ignoradas, tempo de fila por estágio, tempo de processamento por agente, taxa de falhas por motivo, taxa de DLQ e contagem de replays bem-sucedidos. Em ambientes com LLMs, também é recomendável registrar metadados de inferência (modelo, temperatura, top_p, seed quando suportado, e versão do prompt) de forma resumida e sem payload sensível, preservando confidencialidade.

Políticas de retentativa e DLQ orientadas a estado. No DSMO, a retentativa não é uma decisão local de um agente, mas uma política central. Uma implementação robusta distingue falhas transitórias (timeouts, indisponibilidade temporária, rate limit) de falhas permanentes (entrada inválida, contrato quebrado, artefato ausente). Para falhas transitórias, a estratégia típica é backoff exponencial com jitter e limite de tentativas por estágio. Para falhas permanentes, recomenda-se registrar RUN_ABORTED e enviar o contexto para DLQ com metadados suficientes para reprocessamento assistido. O ganho operacional é que o comportamento fica padronizado, mensurável e auditável — e não disperso em “ifs” dentro de agentes heterogêneos.

Snapshot de configuração e versionamento do workflow. Para que replay seja significativo, é essencial capturar a configuração efetiva do run. O campo config_ref deve apontar para um snapshot imutável contendo

parâmetros do pipeline (por exemplo, thresholds, rotas habilitadas, timeouts e limites de retentativa). Da mesma forma, a definição da máquina de estados deve ter um workflow_version que permita reproduzir o grafo e suas condições de transição. O orquestrador deve rejeitar conclusões de estágio cujo workflow_version não corresponda ao run (ou encaminhá-las para uma fila de "inconsistência"), evitando mistura acidental de versões durante deploys.

Armazenamento de artefatos e endereçamento determinístico. Para facilitar replay e auditoria, recomenda-se que o repositório de artefatos use caminhos determinísticos que incorporem run_id, stage, workflow_version e attempt (quando se deseja reter múltiplas tentativas). Exemplo: artifacts/{workflow_version}/{run_id}/{stage}/attempt={n}/output.json.
Mesmo quando se opta por sobrescrever a versão "canônica" do output, é útil manter snapshots por tentativa para permitir análise pós-mortem. Além disso, mensagens de fila devem transportar apenas URIs/refs para entradas e saídas, evitando payloads longos que aumentam custo, latência e risco de truncamento.

Idempotência por estágio e commit transacional. Em ambientes de mensageria, entrega "exatamente uma vez" raramente é garantida. Assim, o DSMO deve operar assumindo redelivery, mensagens duplicadas e reordenação. A estratégia recomendada é implementar um "commit" de estágio no log de eventos com restrição de unicidade por (run_id, stage, workflow_version) e, quando aplicável, por partições lógicas. Ao receber um evento de conclusão, o orquestrador tenta persistir o registro STAGE_SUCCEEDED (ou STAGE_FAILED) de forma atômica; caso o

insert falhe por conflito, o evento é tratado como duplicado e ignorado. Essa prática equivale a um “idempotent receiver” aplicado ao plano de controle, reduzindo trabalho redundante e impedindo que múltiplas conclusões concorrentes avancem o workflow de maneiras divergentes.

Embora o DSMO seja descrito de forma conceitual, sua adoção prática depende de decisões de implementação que preservem o determinismo operacional sob concorrência e falhas. Nesta seção descrevemos um desenho de referência que pode ser implementado com filas duráveis (por exemplo, Pub/Sub, SQS, RabbitMQ ou Azure Storage Queues), um repositório de artefatos (object storage) e um log de eventos append-only. O princípio central é que o orquestrador é o único componente autorizado a materializar transições (enfileirar o próximo estágio) e a “confirmar” efeitos observáveis de um estágio, enquanto agentes apenas consomem mensagens, processam e publicam resultados por referência.

3. CONCLUSÃO

Apresentamos o DSMO, um orquestrador determinístico por máquina de estados para workflows multiagentes baseados em filas, construído sobre uma separação simples de responsabilidades: o orquestrador possui o controle de fluxo e a semântica de auditoria, enquanto os agentes realizam processamento puro por estágio. Esse desenho torna a execução do workflow inspecionável e repetível mesmo na presença de retentativas, concorrência e falhas parciais.

Empiricamente, o DSMO melhora determinismo operacional e confiabilidade em relação à coreografia e ao roteamento orientado por agentes. Ele apresenta maior sucesso de replay e trajetória de estados mais estável (SPS/RSR), elimina processamento órfão e mantém a pressão de DLQ comparável ao baseline de coreografia. Esses ganhos vêm com pequeno aumento na latência mediana e latência de cauda essencialmente inalterada, sugerindo que auditabilidade e determinismo podem ser adicionados sem sacrificar características críticas de desempenho e vazão.

De forma mais ampla, nossos resultados sustentam que a reprodutibilidade em sistemas multiagentes assíncronos é uma propriedade operacional: trajetórias de estado estáveis, efeitos idempotentes e uma trilha imutável de eventos são tão importantes quanto outputs finais estáveis. Trabalhos futuros incluem (i) políticas adaptativas de retentativa e falha informadas por sinais do histórico de eventos e (ii) benchmarking ponta-a-ponta em cargas corporativas de produção e stacks heterogêneas de agentes.

REFERÊNCIAS BIBLIOGRÁFICAS

FOWLER, Martin. Event Sourcing. MartinFowler.com, dez. 2005. Disponível em: <https://martinfowler.com/eaaDev/EventSourcing.html>. Acesso em: 16 fev. 2026.

HOHPE, Gregor; WOOLF, Bobby. Idempotent Receiver. Enterprise Integration Patterns, 2004. Disponível em: <https://www.enterpriseintegrationpatterns.com/patterns/messaging/Idempote
Acesso em: 16 fev. 2026.

NADEEM, Anas; MALIK, Muhammad Zubair. A Case for Microservices Orchestration Using Workflow Engines. arXiv:2204.07210, 2022. Disponível em: <https://arxiv.org/abs/2204.07210>. Acesso em: 16 fev. 2026. DOI: 10.48550/arXiv.2204.07210.