Failure Propagation and Recovery in Agentic Business Operations: A Framework for Reliable Execution

November 2026

Abstract

A business workflow can remain incomplete after an agent has produced a plausible response or received a successful result from one tool. Failures can occur between internal updates, external actions, and the communication of completion. This paper examines how an AI operating system should preserve the state of interrupted work and recover without creating duplicate or contradictory obligations. It develops a recovery framework based on durable operation records, explicit uncertainty, bounded retries, reconciliation, and accountable manual resolution. The analysis draws on idempotent API design and research on agent reliability, then applies the framework to a constructed order cancellation scenario. Its contribution is a distinction between failure detection, containment, and verified business recovery, together with measures for unresolved work. MatruOS provides an architectural context. The paper proposes tests rather than reporting experimental results. It concludes that reliable execution requires ownership of the work remaining after an error, including effects that cannot be automatically reversed.

Keywords: agent reliability, failure recovery, reconciliation, idempotency, unresolved operations, business continuity.

1. When a failed response leaves a real obligation

A customer requests cancellation of an order. The agent updates the internal order status, asks a courier to cancel collection, and submits a refund request. The courier response is delayed and the refund response times out. The agent cannot establish whether the shipment will still be collected or whether the refund was processed.

If the system reports cancellation as complete, the customer may receive goods after being refunded. If it repeats the entire workflow, it may issue a duplicate refund. If it abandons the task, an employee inherits an obligation without a clear record of what happened. Each outcome arises from treating a partial workflow as one undifferentiated success or failure.

The research question is how an AI operating system should represent and recover work when only some actions have completed. The unit of analysis is the business operation from its initial request to a verified terminal state. Recovery includes human intervention where automatic correction is inappropriate or unavailable.

The scope differs from initial transaction integrity. Transaction controls aim to prevent invalid changes within their boundary. Recovery examines the period after an interruption, including uncertainty about external effects and obligations that continue despite a software error.

2. Research basis and method

Featonby’s discussion of idempotent APIs explains how a stable request identifier can allow repeated attempts to represent the same intended effect. The contract must distinguish repeated intent from a new request and

account for delayed arrivals. This supports recovery from uncertain responses within the service's stated semantics. [1]

Stripe's API documentation provides a concrete example of why those semantics must be read carefully. Its API v1 records the initial response after execution begins, including error responses, and its key retention rules affect later reuse. Recovery logic must follow the actual integration contract rather than a generic assumption about retries. [2]

Rabanser and colleagues examine reliability through consistency, robustness, predictability, and safety. Their framework supports assessing how agents fail and behave under variation instead of compressing performance into one success rate. The present paper applies that orientation to the unfinished business obligations that remain after a failure. [3]

The method is conceptual synthesis and failure analysis of constructed workflows. The proposed contribution is an operational recovery record and an evaluation design for uncertainty resolution. The paper does not claim to introduce idempotency or fault tolerance, and it reports no measured recovery performance.

3. A taxonomy of operational failure

3.1 Interpretation and planning failures

An agent may misunderstand the customer, choose the wrong record, or omit a necessary dependency. Recovery begins by identifying whether the planned action was valid. Repeating an invalid plan faithfully can reproduce the same harm.

For example, a cancellation request may apply to one line of an order rather than the whole order. If the agent cancels every line, the corrective workflow must identify which commitments remain valid. The original conversation and the interpreted scope should be retained so that the difference can be investigated.

Planning failures should therefore be separated from execution failures. A service outage and an incorrect customer identity need different remedies. The system should expose the relevant category to the person responsible for resolution.

3.2 Execution failures with known outcomes

Some failures have a definite outcome. A validation error may establish that an order was rejected before mutation. A confirmed inventory update may establish that a reservation exists. Recovery can proceed from those known states using the service's documented response.

Even then, the application should verify which part of the workflow the response covers. A successful submission may mean that a request was accepted for processing, not that the requested action is complete. The operation record should preserve both acceptance and completion where they are distinct.

Known failure can lead to a revised plan, but revision needs the appropriate authority. A rejected supplier booking does not automatically permit the agent to select a different supplier with different terms. The recovery process should remain within the original delegation or seek a new approval.

3.3 Uncertain outcomes

A timeout is evidence that the response was not received within the expected interval. It does not establish that the external action failed. The request may have been processed, rejected, or left pending. The system should record uncertainty explicitly and use available query or reconciliation mechanisms.

A new request with a new identifier can create a second effect if the first succeeded. A permanent refusal to retry can also leave legitimate work incomplete. The appropriate procedure depends on the provider's contract and the business consequence of duplication.

Uncertainty should have an owner and an age. A queue of unresolved operations is an operational liability that requires attention. It should not disappear from reporting because the agent's conversational session has ended.

3.4 Business failures after technical success

An action can complete technically and still fail the business purpose. A refund may reach the wrong transaction, a shipment may use an unsuitable destination, or a cancellation may occur after a supplier's permitted deadline. The system needs validation of the business result in addition to API status.

This category is important because monitoring based only on error codes can report healthy execution while obligations are incorrect. Reconciliation should compare the observed state with the intended and authorised outcome. The agent's own summary is one input, not the reference result.

4. A durable record of unfinished work

The operation record should connect the original intent, transaction version, authority, planned steps, attempts, provider identifiers, and observed outcomes. Each step needs a status that distinguishes not attempted, accepted, completed, rejected, and uncertain where those distinctions apply.

The record should also identify dependencies. A refund may depend on confirmation of cancellation, or the business may permit an immediate refund while separately managing goods recovery. The policy determines that relationship. The agent should not infer it solely from the order in which tools are available.

Durability matters across restarts and handoffs. If a worker stops, another worker or authorised employee should be able to determine what is safe to do next. Reconstructing the workflow from a long conversation alone creates avoidable ambiguity, particularly when several attempts used different plans.

The operation should retain a deadline or escalation condition appropriate to its consequence. A pending customer message and an uncertain payment can require different response times. A generic retry queue cannot express every business priority.

The record also needs a closure criterion. Manual resolution should identify the evidence and person who closed the case. Marking an issue "handled" without recording the final state can leave later staff unable to distinguish a completed correction from an abandoned investigation.

5. Recovery mechanisms and their limits

5.1 Bounded retries

Retries should be limited by an explicit policy covering attempt count, delay, elapsed time, and the kind of error. Repeatedly invoking an unavailable service can increase load and postpone useful escalation. The policy should reflect whether the operation is safely repeatable and whether its outcome is already uncertain.

The request identity should persist across retries of the same intent. If the parameters change materially, the system should treat the change as a revised operation with the necessary approval and relationship to the original. A fresh identifier should not be used merely to bypass an error response.

A retry can also encounter changed business conditions. Stock may have been allocated elsewhere or authority may have expired. The service should revalidate applicable conditions while preserving the identity of the original attempt. Recovery does not suspend governance.

5.2 Reconciliation before repetition

Where the provider supports querying an operation, reconciliation should attempt to establish the external state using its identifier and evidence. A confirmed external result can resolve uncertainty without repeating the effect. If the provider cannot establish the state, the workflow may require manual investigation.

Reconciliation should be independent enough to detect disagreement. Reading the same local flag that the failed worker wrote does not verify an external refund. The check should inspect the relevant provider or authoritative business record.

The process should also account for delayed visibility. An external action may be accepted but not immediately queryable. The recovery policy should define how long to wait, when to retry the query, and when to escalate. These values need testing against the integration’s actual behaviour.

5.3 Compensation

Compensation creates a new action intended to address an earlier effect. It may release a reservation, cancel a booking, or issue an authorised correction. It should preserve the original event and record the relationship between the two.

Some effects are only partially compensable. A message already read cannot be made unread. A dispatched shipment may require return logistics. A refund may incur fees or create accounting consequences. The recovery model should identify these limits before execution so that the system can choose appropriate approval and sequencing.

Compensation can fail. The workflow should therefore track it as work requiring verification, rather than assuming that calling a compensating tool restores the original state. The remaining obligation should stay visible until it is resolved or formally accepted by an authorised person.

5.4 Controlled manual resolution

Manual intervention needs a clear handoff containing the intended outcome, known effects, uncertainty, and permitted next steps. An employee should not have to infer whether pressing a retry button may duplicate a payment.

The interface should prevent simultaneous automated recovery while a person is resolving the same case, or coordinate them through an explicit ownership mechanism. Otherwise, two well intentioned interventions can create contradictory changes. Taking ownership and releasing it should be recorded.

Manual resolution should feed back into the system. A recurring exception may indicate an integration defect, an ambiguous rule, or missing provider functionality. Recording the cause allows targeted improvement rather than repeated individual correction.

6. Worked cancellation scenario

Assume an order has been paid and scheduled for courier collection. The customer requests cancellation before dispatch. Policy requires confirmation that collection can be stopped before the refund is finalised. The agent records the request and obtains the permitted authority.

The courier accepts the cancellation request but reports processing as pending. The operation remains open. The agent can tell the customer that cancellation is being processed, with a clear account of what remains unresolved. It should not state that the refund is complete.

The next provider query confirms cancellation, and the refund operation is submitted with a stable identifier. Its response times out. The worker records the uncertain outcome and queries the payment provider. If the refund is confirmed, the workflow records completion and sends the customer notification.

If the provider remains unavailable, an authorised employee receives the case with the refund identifier and known status. The system prevents a new refund request from being submitted as if no attempt existed. The employee may investigate through another authorised channel or continue waiting according to the recovery policy.

Suppose the courier later reports that goods were collected despite cancellation confirmation. The system must open a new exception linked to the original operation. It cannot rewrite history to claim that cancellation never occurred. The business now has a goods recovery problem, and the record should show the provider’s earlier confirmation and subsequent discrepancy.

This scenario demonstrates why recovery extends beyond the first apparent terminal state. Some obligations require later reconciliation or monitoring. The design should specify the period and evidence needed for that follow up.

7. Application to MatruOS

A MatruOS deployment should connect agent tasks to durable business operation records that remain available after the conversation ends. Order, payment, and communication states should be separately visible where their outcomes differ. The interface should explain the status in business language while preserving technical identifiers for investigation.

The recovery service should operate through the same permission controls as ordinary execution. An error handler should not become an unrestricted administrative route. Actions taken during recovery can be as consequential as the original operation and require corresponding authority.

Operational ownership is essential. Each category of unresolved work needs a responsible role, escalation route, and closure procedure. A deployment should establish who monitors those queues outside the initial project team. Reliability depends on maintenance and support arrangements as well as software behaviour.

The requirements here are proposed for architecture review and testing. No claim is made about measured MatruOS recovery time or production incident frequency. Those results would require observed deployments and a documented evaluation period.

8. Proposed evaluation protocol

Inject failures at each boundary: before a request is sent, after acceptance, after an external effect, and before the local record receives confirmation. Include duplicate responses, delayed callbacks, worker restarts, and changes in authority during recovery. Use synthetic records and controlled services to avoid real obligations.

Measure duplicate effects, incorrect completion reports, unresolved operation age, time to verified recovery, and manual effort. Report the distribution of recovery times, including the longest unresolved cases. An average calculated only from completed recoveries can hide abandoned work.

The reference outcome should be determined from authoritative service state and business policy. The evaluator should inspect all participating systems after delayed events have arrived. A run is not fully assessed when the agent stops responding.

The principal proposition is that explicit uncertainty and durable operation identity reduce unsafe repetition after ambiguous failures. A second proposition is that structured handoff reduces investigation effort without increasing incorrect closure. A third is that failure categories predict different recovery costs and should therefore be reported separately.

Repeated trials should vary timing and concurrency. A successful recovery under one event order may fail under another. Preserve the test configuration, model version, provider simulation, and operation traces so that the observed result can be reproduced and investigated.

9. Limitations and conclusion

Recovery depends on the information and controls offered by external services. Some providers cannot query an operation reliably or reverse its effect. The architecture should make those limits explicit and narrow delegation accordingly.

The framework also depends on available operational staff. Escalation is meaningful only when someone can investigate and act. A deployment that produces unresolved work beyond the organisation’s capacity needs a change in scope, integration design, or support arrangements.

Reliable agentic execution requires a maintained account of unfinished work. Detection identifies a problem, containment prevents further unsupported effects, and recovery establishes the final business state. These are distinct responsibilities that should be evaluated separately.

An AI operating system should preserve operation identity, uncertainty, and ownership until the obligation is resolved. Its reliability is demonstrated through the handling of interrupted and disputed work as well as ordinary completion.

Author affiliation and disclosure

The author is affiliated with Code Colonies Private Limited, Ahmedabad, and leads the MatruOS team. MatruOS is an architectural application context. No independent product evaluation is reported.

References

[1] Featonby, M. Making retries safe with idempotent APIs. Amazon Builders’ Library. Accessed 2 November 2026.

https://aws.amazon.com/builders-library/making-retries-safe-with-idempotent-APIs/

[2] Stripe. Idempotent requests. API documentation. Accessed 2 November 2026.

https://docs.stripe.com/api/idempotent_requests

[3] Rabanser, S., Kapoor, S., Kirgis, P., Liu, K., Utpala, S., and Narayanan, A. (2026). Towards a Science of AI Agent Reliability. Proceedings of the 43rd International Conference on Machine Learning.

https://proceedings.mlr.press/v306/rabanser26a.html