Enterprise Transition to an AI Operating System: Interoperability, Context Portability, and Operational Continuity

November 2026

Abstract

Introducing an AI operating system into an established enterprise requires decisions about which system owns each record and which service may create a business commitment during transition. This paper develops a migration framework that connects interoperability, data reconciliation, authority transfer, and recovery. It draws on incremental modernisation, change data capture, and provenance concepts, then examines a constructed transition of an order workflow. Its contribution is an explicit handover record for business authority, including the evidence needed to move from observation to bounded execution and the limits of rollback after external effects. The paper also examines context portability when models or platform providers change. It proposes evaluation criteria for continued operation during migration, including duplicate writes, missing updates, unresolved obligations, and staff access to the current state. MatruOS provides an architectural context. No migration case study or measured deployment outcome is reported.

Keywords: enterprise migration, interoperability, authority transfer, reconciliation, context portability, operational continuity.

1. Migration changes who may act

An enterprise may want an AI assistant to prepare and execute orders while its existing enterprise resource planning system continues to manage inventory and finance. The initial integration can appear straightforward: read the records, generate a proposal, and write the accepted order. The difficult questions arise when both old and new workflows remain active.

Which system owns the order number? Which service reserves stock? What happens if an employee edits the order in the existing application after the agent has prepared a plan? If the new service is paused, can staff identify every commitment it created? These questions concern operational authority as much as data movement.

The research question is how an enterprise can introduce an AI operating system while preserving consistent records and continuing business obligations. The unit of analysis is a workflow moving between operating arrangements. The paper considers both integration with retained systems and gradual replacement of selected functionality.

A successful transition should leave the business able to identify the current state, the authorised writer, and the recovery procedure at every stage. Installing a new interface or copying historical data does not by itself establish those conditions.

2. Research foundations and method

Fowler’s account of the Strangler Fig approach describes gradual modernisation of existing applications. It supports migration through bounded changes rather than assuming that all functionality must move at once. The appropriate boundary still requires analysis of the business and its dependencies. [1]

Debezium’s architecture documentation describes change data capture from source databases and propagation through event infrastructure. This provides a mechanism for observing record changes. The availability of a change stream does not establish that the receiving system has the same business meaning or authority as the source. [2]

The W3C provenance model supplies concepts for recording relationships among information, activities, and responsible actors. These concepts can support traceability of transformed records during migration. Additional business rules are needed to decide whether a transformed record is fit for operational use. [3]

The present paper uses these foundations to develop a transition protocol centred on record ownership and business commitments. It is a conceptual design with constructed examples. It does not claim a universal migration sequence, a guaranteed duration, or empirical validation of the proposed protocol.

3. Establishing the source of authority

3.1 Record ownership

For each relevant entity, the migration plan should identify the authoritative system and the operations it permits. Customer identity, price terms, stock, and invoices may have different owners. A general statement that one platform is the “single source of truth” can conceal these distinctions.

Ownership should be defined at the level required by the workflow. One system may own customer contact details while another owns approved credit terms. If that division exists, the integration contract should describe it explicitly. The new agent service should not overwrite a protected field merely because it can update another part of the record.

The plan should also identify who resolves discrepancies. A connector can detect that two records disagree, but deciding which business value is correct may require a responsible employee. Unresolved differences should be recorded and excluded from consequential execution where appropriate.

3.2 Identifiers and semantic mapping

The same customer may have different identifiers across sales, finance, and commerce systems. Mapping needs to account for duplicates, merged accounts, and historical changes. Matching names alone can produce incorrect associations, particularly where several branches or legal entities share similar names.

Field mapping also requires meaning. “Available quantity” may include expected receipts in one system and exclude them in another. “Paid” may mean payment received, allocated, or settled. A migration should document those differences and decide which interpretation governs the selected workflow.

The evidence should include examples covering ordinary and exceptional records. A mapping that works for a standard order may fail for partial delivery, credit notes, or returned goods. Testing these cases before authority transfer prevents the agent from acting on a coherent but incorrect interpretation.

3.3 Data history and current state

Historical records can support analysis and context, while current records support action. The migration plan should distinguish those uses. A historical invoice with an obsolete product code may remain useful for reference without being eligible to create a new order.

Transformations should preserve lineage and uncertainty. If a field is inferred or normalised, the receiving system should retain enough information to explain the change. Missing values should not be silently replaced with convenient defaults that later appear authoritative.

A clear starting point is needed for current operation. The enterprise should know which records were reconciled, which changes occurred afterwards, and how those changes reached the receiving system. This establishes the basis for evaluating freshness and completeness.

4. A staged transition of responsibility

4.1 Observation

The new system can initially read records and produce proposals without creating business effects. This allows the team to examine entity mapping, retrieval, and interpretation under ordinary workload. The output should remain clearly separated from actual commitments.

Observation should include disagreement analysis. When the new system’s proposal differs from the existing process, reviewers should determine whether the difference reflects an error, an ambiguous rule, or a legitimate alternative. Automatically treating the existing output as correct can reproduce existing defects.

This stage should produce a record of unresolved issues and the conditions required before execution is permitted. It should not continue indefinitely without a decision about scope. Observation is useful when it resolves specific uncertainties.

4.2 Assisted execution

The next stage can allow preparation of actions for human review. The approved request is executed through a controlled interface to the authoritative system. This tests the relationship between proposal, approval, and actual state change while retaining a bounded delegation.

The reviewer should see the material fields and the version of the record on which the proposal depends. If those fields change before execution, the service should revalidate the request. A review conducted against a stale proposal should not silently authorise a changed transaction.

Assisted execution also reveals operational support needs. The team should record rejected writes, unavailable services, unclear responses, and the effort required to resolve them. These observations inform whether the integration is ready for further responsibility.

4.3 Bounded delegation

Delegation can expand when the workflow has stable definitions, dependable interfaces, and manageable exception handling. The scope should identify permitted actions, limits, and conditions that trigger human review. Expansion should be a recorded decision by the responsible business owner.

The old and new paths must have an explicit coordination arrangement. They may both remain available, but they should use common services or other controls that preserve the same constraints. Independent writes to overlapping state without coordination can create duplication or conflicting commitments.

A defined handover point is useful where ownership changes. The record should state when the new service becomes the authorised writer, which pending operations remain with the old service, and how delayed messages are handled. The transition should be observable rather than inferred from a deployment timestamp.

5. Synchronisation and reconciliation

Change data capture can propagate updates from an existing database, but the receiving system must account for lag, duplicates, and the meaning of captured changes. [2] A current looking interface may display data that has not yet incorporated the latest source event. Consequential actions should use a freshness rule appropriate to the workflow.

The integration should preserve source identifiers and relevant version or ordering information. A delayed update should not overwrite a newer state. Where events from several sources must be combined, the service needs an explicit method for detecting incomplete or conflicting views.

Reconciliation should inspect both counts and business meaning. Matching the number of orders does not establish that quantities, customers, and commitments match. Aggregate totals can also conceal offsetting errors. The process should combine summary checks with record-level validation of consequential fields.

Differences require classification. Some arise from expected timing, others from transformation errors, missed events, or legitimate changes made through another path. Each category needs a resolution procedure and owner. A dashboard that merely displays a growing discrepancy count does not maintain continuity.

The enterprise should also test recovery after an interruption in the change stream. The receiving system needs a way to establish which updates were missed and whether a fresh snapshot is required. Replaying events must respect the receiving service’s deduplication and ordering rules.

6. A constructed order transition

Assume an enterprise retains its existing ERP for stock and invoicing while introducing an AI operating system for customer enquiries and order preparation. Initially, the agent reads approved customer terms and current stock to prepare a proposal. An employee enters accepted orders through the existing process.

During observation, the team discovers that the ERP’s stock value includes items under quality inspection. The receiving system had treated them as available for commitment. The mapping is corrected to use allocatable stock, and affected test proposals are reviewed. This resolves a semantic issue before execution authority expands.

In assisted execution, approved proposals are submitted through an order service that assigns the authoritative order identifier and reserves stock. The service returns a durable status. If its response is uncertain, the new system queries the operation rather than creating a second order.

For bounded delegation, the enterprise permits standard orders within defined terms to proceed without individual review. Exceptional pricing and disputed customer balances remain subject to approval. Employees can still use the existing interface, but both paths call services that enforce the same resource constraints.

Suppose a deployment defect requires pausing the new workflow. New requests return to the existing process, while already committed orders remain visible in the ERP. Pending proposals are identified separately from transmitted orders. The recovery plan assigns responsibility for every uncertain submission before the new service is restarted.

The scenario illustrates a proposed handover record: scope, authoritative writer, pending work, reconciliation status, and recovery owner. The record makes it possible to explain what moved and what remains under the previous arrangement.

7. Rollback after business effects

Rolling back application code does not reverse orders, messages, or payments already created. The migration plan should distinguish software rollback from business recovery. Each consequential action needs an account of how it will be retained, corrected, or reconciled if the new service is withdrawn.

A return to the old interface can be safe only if it has access to the current obligations. If the new platform created records unavailable to the old system, an immediate fallback may leave staff operating from an incomplete view. The transition plan should address that dependency before live execution begins.

Pending work requires particular attention. A request may be queued, transmitted, accepted, or completed at the moment of rollback. Treating every pending item as unprocessed can produce duplicates. The recovery team needs operation identifiers and evidence of external effects.

The rollback rehearsal should therefore include real workflow states in a controlled environment. The team should demonstrate that it can stop new actions, identify uncertain work, restore an authorised operating path, and verify the remaining obligations. A successful application restart is only one part of that exercise.

8. Context portability and model changes

Portability concerns the ability to retain usable business context when technology changes. Records should be exportable with identifiers, relationships, policy versions, and provenance. A collection of document files may omit the links that made those documents operationally meaningful.

The export should also preserve permissions or provide a reviewed mapping to the receiving system’s access model. Moving content without its access conditions can create disclosure. Conversely, an unusable export can leave the business dependent on a provider despite nominal ownership of its files.

Model replacement requires behavioural evaluation. The same context can produce different tool choices, interpretations, or uncertainty handling under another model. The transition should preserve representative test cases and compare the required outcomes before changing the production configuration.

Portability should include operational procedures. Staff need to know how to retrieve records, inspect pending work, and continue essential tasks if a provider becomes unavailable. An export feature that has never been exercised provides limited evidence of continuity.

9. Application to MatruOS and evaluation design

For MatruOS, the framework supports deployment as either a bounded integration layer or a broader business environment, depending on the enterprise’s existing systems. The choice should follow record ownership and workflow requirements. It should not be determined solely by a preference for consolidation.

The proposed evaluation should measure duplicate writes, missing updates, stale decisions, reconciliation effort, and time to restore a valid operating path. It should also assess whether staff can identify the authoritative state during transition. Technical consistency can coexist with operational confusion if the interface does not communicate ownership clearly.

Test scenarios should include delayed events, changed schemas, merged customers, partial migration, connector outages, and a rollback while requests are in progress. Each scenario needs expected business outcomes and evidence from all participating systems. The agent’s report should be checked against those observations.

The principal research proposition is that explicit authority handover and pending work reconciliation reduce transition ambiguity. A second is that semantic mapping errors create risks not detected by simple record counts. A third is that portability requires behavioural and operational testing as well as data export. These propositions remain to be evaluated in deployment studies.

10. Limitations and conclusion

The framework assumes access to interfaces and records sufficient for reconciliation. Some existing systems restrict exports, lack stable identifiers, or expose limited status information. These constraints may require a narrower scope or additional preparation before delegation is appropriate.

Migration also depends on organisational cooperation. Different departments may disagree about ownership or the meaning of a field. The implementation should record and resolve those disagreements rather than treating them as connector configuration issues.

An enterprise transition to an AI operating system is a transfer of operational responsibility supported by data and interfaces. The proposed framework keeps record ownership, authority, pending work, and recovery visible throughout that transfer.

Continuity is established when the business can keep working from a trustworthy account of its obligations, including during interruption or withdrawal of the new service. That is the appropriate basis for deciding when a workflow is ready to move and how far responsibility should expand.

Author affiliation and disclosure

The author is affiliated with Code Colonies Private Limited, Ahmedabad, and leads the MatruOS team. The order migration is a constructed scenario. No independent deployment evaluation is reported.

References

[1] Fowler, M. (2024). Strangler Fig. 22 August.

https://martinfowler.com/bliki/StranglerFigApplication.html

[2] Debezium Community. Debezium Architecture. Reference documentation. Accessed 2 November 2026.

https://debezium.io/documentation/reference/stable/architecture.html

[3] Moreau, L., and Missier, P., editors (2013). PROV-DM: The PROV Data Model. W3C Recommendation, 30 April.

https://www.w3.org/TR/prov-dm/