RATHEESH TECHNOLOGY LTD.
AI IMPLEMENTATION & GOVERNANCE

2026

Case Study 01

ENTERPRISE DENTAL CLAIMS HYPERAUTOMATION

Author:
Ratheesh Ramadasan

Re-architecting a high-risk, multi-location money-out process for scale, resilience, maintainability and zero-defect production

368-599-2299 ai.rrlabs.ca ratheesh.ramadasan@gmail.com

1. Executive Summary

This case study describes the re-architecture of a high-risk, high-volume dental claims operation that had to process current BAU work and several months of backlog across five operating locations. The work was financially consequential because the process ultimately resulted in claim submission and potential money-out execution. The transformation therefore had to be treated as a controlled production operating model, not as a simple bot build.

The solution separated orchestration, rules, execution, security, infrastructure, exception handling, reporting and operational ownership. A SQL-backed control plane governed transaction state, locks, retries, rule data, historical information, SLA timing and reporting. Eighteen parallel bots executed user-like work against a third-party claims platform that did not offer a suitable API or a dedicated UAT path.

The final architecture improved throughput, reduced backlog, protected BAU processing, enabled self-healing recovery, used CyberArk for secure credential retrieval, established controlled live-system validation and kept human judgment in the loop for ambiguous cases. The result demonstrated true HyperAutomation: people, process, data, technology, security, operating model and production control working together.

2. Business Context and Operational Risk

Dental Claims was operated from five locations and included high seasonal volume, historical dependencies and complex adjudication rules. The processing logic could depend on historical claims, provisional conditions, deduction percentages, eligibility rules, government rules, previous processing decisions and judgment-based exceptions. The rule space was not small; more than 10,000 possible scenario combinations had to be considered.

At program entry, several streams carried approximately two to six months of backlog. The backlog was not merely a productivity concern. Since this was a money-out process, unprocessed claims represented outstanding obligations, customer-service risk, audit exposure, operational-control risk and reputational pressure.

- Five processing locations contributed work into a shared operational challenge.
- Current BAU and historical backlog had to be managed in parallel.
- Backlog remediation could not disrupt incoming daily claims.
- Financial accuracy, auditability and end-of-day cut-off controls were non-negotiable.

3. Existing-State Pain Points

The existing automation was not positioned to scale. It relied on multiple technologies, had many failure points, lacked a reliable requirements baseline and could process only a small portion of total demand. Incremental repairs would have preserved fragility rather than creating an enterprise production service.

The third-party claims platform had no suitable API. Automation therefore needed to respect user-like interaction speed, application constraints and licensing limits. Throughput could not be solved by pushing one bot faster; it required controlled horizontal scale and strong state management.

- Only approximately 8-10% of the required volume was being processed by the existing automation.
- The implementation contained multiple points of failure and was difficult to support.
- No reliable requirements document existed from which to rebuild confidently.
- No suitable API existed for the external claims platform.

- No fit-for-purpose UAT environment existed for full end-to-end testing.
- Licensing and application constraints limited how automation could interact with the system.

4. Key Architecture Decision - Rebuild From First Principles

The critical architecture decision was not to reverse-engineer failed automation and treat its code as the business requirement. The better path was to reconstruct the requirement from the live operation, validate rules and exceptions with Operations SMEs, and design a new control architecture that could withstand production scale.

This decision changed the engagement from bot repair to process re-architecture. The bot became only one worker type in a managed processing service. The control plane became the centre of the design.

5. Dual-Track Recovery Model

The backlog could not be remediated by diverting all capacity away from BAU work. That would simply create a new backlog. The operating model therefore separated current processing from historical remediation while allowing both streams to use the same controlled architecture.

Dedicated backlog processing capacity was created while BAU claims continued. Work was prioritized and governed centrally, allowing the program to reduce outstanding inventory without sacrificing the current operating rhythm.

- Track A: BAU processing for incoming current claims.
- Track B: backlog remediation for historical inventory.
- Both tracks used controlled intake, state tracking, exception handling and reporting.
- Backlog remediation was processed and reviewed in approximately one month while current BAU continued.

6. SME-Led Requirements Reconstruction

Because there was no reliable end-to-end requirements document, requirements were reconstructed through daily collaboration with Operations SMEs. The team captured the business rule, identified the smallest testable unit, created supporting data scenarios, added the requirement to the build backlog and validated the result with Operations.

The working rhythm mattered. Operations SMEs brought practical knowledge from the claims floor, including exception comments, historical dependencies and scenarios that were not visible in system fields alone. This created operationally accurate requirements rather than technically convenient automation assumptions.

- SMEs identified the day's business rule or scenario.
- The team decomposed it into a smallest testable increment.
- Relevant claims and data scenarios were identified.
- The build was updated and defects from the prior cycle were corrected.
- Operations validated the result the next day.
- The cycle repeated until the solution was aligned with operational reality.

7. Agile Delivery Without Architecture-by-Iteration

Agile delivery did not mean the architecture was allowed to emerge randomly. The control framework, risk boundaries, production constraints and target architecture were established first. Detailed claims logic was then elaborated through small, validated increments.

This distinction was important. For a money-out process, core controls such as duplicate prevention, credential security, transaction locking, state persistence, auditability and financial cut-off handling had to be designed up front. Only the detailed rule scenarios evolved iteratively.

8. Controlled Live-System Testing Without UAT

The third-party claims platform did not provide a suitable UAT environment for complete automation testing. The only realistic end-to-end validation path involved the live system. This introduced significant risk because final submission could result in financial processing.

The program created a controlled production-validation approach. Specific claims were reserved and locked for bot testing so they could not be processed simultaneously by a human and the bot. Authority was increased progressively as evidence and confidence increased.

- Phase I - Save Only: the bot processed and saved the claim but did not submit it.
- Operations immediately validated the saved work and provided feedback.
- Phase II - Controlled Submission: specifically identified claims could be submitted after confidence improved.
- Operations immediately validated submitted items before end-of-day financial processing.
- If a defect was found in the validation window, Operations could revoke the request before financial execution.
- Two validation incidents were detected and contained before financial processing; no defective claim reached financial execution.

9. Target Architecture Overview

The redesigned target state deliberately separated intake, orchestration, rules, allocation, execution, security, application interaction, human controls and reporting. This made the solution resilient to scale, rule changes and operational exceptions.

The SQL control layer was the operational brain. Bots did not independently decide what to process next. They acquired eligible work from the queue, processed it, recorded outcome/state, released locks and moved to the next item.

Architecture Layer Responsibilities

Layer	Responsibility	Why It Mattered
Claims Intake	Accept current BAU and backlog work from five locations.	Created a central view of demand and avoided unmanaged parallel processing.
SQL Control Plane	Queue, locks, state, rules, calculations, historical data, retries, SLA timing, audit and reporting.	Separated orchestration from bot execution and prevented duplicate processing.
Work Allocation	Select eligible transactions based on priority, readiness, lock state and SLA.	Ensured available bots processed the right work at the right time.
Parallel Bot Worker Pool	Eighteen bots processed work concurrently.	Throughput came from horizontal scale at user-like application speed.
Terminal Server Infrastructure	Supported multiple concurrent sessions.	Reduced one-bot-one-VM cost and operational overhead.
CyberArk Security	Controlled credential retrieval; no credentials in code.	Reduced credential exposure and supported secure multi-session operation.

Third-Party Platform	Live UI interaction against external claims application.	Respected no-API, no-UAT and license constraints.
Operations Control	Handled ambiguity, validation, revocation windows, exceptions and reporting.	Kept judgment and financial safety in the operating model.

10. SQL-Backed Control Plane and Custom Queue

A SQL database became the control plane because the program required queue and multi-bot orchestration behaviours beyond simple bot work queues. It provided a single authoritative state for every claim transaction.

The SQL model captured transaction ownership, processing state, retry status, historical context, rule values, calculations, exception reason codes, processing time and audit/reporting information. Bots became stateless workers using the control plane rather than isolated automations holding business context locally.

- Central transaction ownership and locking prevented duplicate processing across parallel workers.
- Processing state, retry and completion information were persisted centrally.
- Historical claim data and calculations could be accessed without repeatedly traversing UI screens.
- The queue allowed available bots to pull eligible work in a controlled manner.
- Operations and support teams could understand where each claim stood at any point in the day.

11. Configuration-Driven Rule Architecture

Business and government rules changed over time. Embedding all rules and calculations inside monolithic bot code would have made the solution fragile and expensive to maintain. The design externalized volatile rules, values and calculations into SQL tables, queries and configuration structures where appropriate.

The pattern was stable execution logic plus configurable business rules. This made changes easier to test, audit and deploy without distributing logic across many bot workflows.

12. Business Rule Complexity and Deterministic Processing

The solution did not attempt to force all claims through automation. It distinguished deterministic processing from cases requiring ambiguity resolution or judgment. Deterministic claims could proceed through validation, adjudication, deduction calculation, rule application and submission. Non-deterministic claims preserved work and context for Operations.

This was a deliberate control decision. A lower STP rate with zero processing defects was more valuable than a higher automation percentage that created financial or audit risk.

13. Parallel Bot Worker Pool

Approximately eighteen bots operated in parallel. Each bot acquired the next eligible transaction, processed the claim through the third-party UI, recorded state and either completed or transferred the claim based on deterministic outcome.

The worker model avoided fixed ownership of a backlog segment by a single bot. If a worker became unavailable, the control layer could release the transaction and allow another worker to continue.

14. Terminal Server Execution Infrastructure

Because the claims application required user-like interaction, the architecture scaled execution through controlled multi-session infrastructure. Terminal Server capacity supported approximately twenty concurrent sessions under typical load and could support up to roughly thirty-five depending on workload and environmental constraints.

This avoided a linear one-bot-one-VM model, reduced infrastructure footprint and made operational capacity more practical.

15. Security and CyberArk Credential Management

Application credentials were separated from automation code and retrieved through CyberArk under controlled access. This reduced credential exposure, avoided hard-coded passwords and enabled a secure operating model across parallel unattended sessions.

Credential control was not treated as a late-stage technical detail. It was part of the production architecture because bot identities represented access to financial processing capability.

16. Duplicate Processing Control

Duplicate processing was one of the most important risks. Multiple bots working in parallel could not be allowed to process the same claim. The SQL control plane therefore used transaction locking and ownership controls so each eligible claim was acquired by one worker at a time.

Locks also supported controlled live testing. Reserved claims could be protected from simultaneous manual and bot processing, which was critical because the live system had no separate UAT path.

17. Self-Healing Recovery

Every active transaction had an expected processing window. The control layer monitored claims that remained locked or active beyond configured duration. When timeout conditions were met, support logic could terminate or recover the affected bot/session, release the lock and return the work item to the queue.

A stalled bot therefore did not permanently stall the business request. This created a self-healing recovery pattern across the worker pool.

- Detect abnormal processing duration.
- Review processing/log state.
- Terminate or recover the affected bot/session when appropriate.
- Release the transaction lock.
- Requeue the item for another worker.
- Preserve audit and recovery evidence.

18. End-of-Day SLA and 6 PM Capacity Control

The operation had a daily cut-off around 6 PM. The architecture monitored remaining work, available capacity and expected throughput. If remaining demand exceeded projected capacity, alerts could be triggered and unprocessed items could be moved to appropriate pending/manual states.

This converted end-of-day risk from surprise discovery into managed operational control. The solution did not simply run bots until someone noticed a problem.

19. Human Continuation and Exception State Model

Roughly forty percent of volume required human continuation because ambiguity, judgment or non-deterministic conditions were present. The architecture preserved completed work, state and reason codes so Operations could continue without losing context.

Manual handoff was designed into the process rather than treated as bot failure.

Exception State Model

State	Meaning	Control Purpose
Completed - STP	Claim completed end to end through deterministic automation.	Straight-through completion with audit evidence.
Manual Review	Ambiguity or judgment required; work and context preserved.	Keeps humans in control where business judgment is required.
Requeued	Worker or session exceeded allowed processing time or became unavailable.	Supports recovery without duplicate processing.
Pending at Cut-Off	Not completed before daily processing deadline.	Supports end-of-day governance and BAU continuity.
Validation Hold	Reserved/controlled claim awaiting Operations confirmation.	Controls live-system testing risk.

20. Automated Reporting and Operational Dashboarding

Once workload was controlled, reporting became a natural output of the architecture. The SQL control plane could support reports on claims processed, claims pending, exceptions, bot throughput, SLA risk, capacity needs and daily completion.

This converted automation from a black box into an observable production service.

21. CoE and Production Operating Model

The transformation established more than a technical solution. A structured Automation Center of Excellence and production support model was created around architecture, standards, testing, controlled release, monitoring, support, maintenance and change.

This operating model was necessary because the solution interacted with live financial processing. It needed ownership across Operations, RPA, infrastructure, security and support.

- Architecture and reusable implementation standards.
- Automated deployment and controlled release practices.
- Bot health and transaction health monitoring.
- Incident handling and support ownership.
- Business-rule and application-change maintenance.
- Credential and infrastructure-capacity ownership.

22. Risk and Control Traceability

Risk / Requirement / Control Traceability

Risk or Requirement	Architecture / Control Response	Operational Evidence
Multi-month backlog and audit/corporate risk	Parallel backlog remediation while protecting BAU processing.	Backlog processed/reviewed in approximately one month.
No reliable requirements baseline	SME-led reconstruction from actual business process.	Daily requirements and validation loop.
No UAT environment	Controlled live-system validation with reserved claims.	Two incidents contained before financial processing.
Financial submission risk	Progressive authority from save-only to controlled submission.	Operations validation and revocation window.
10,000+ rule combinations	Small testable increments and configuration-driven rules.	Improved maintainability and testability.
Duplicate processing	SQL transaction ownership and locks.	One bot per claim transaction.
Bot/session stall	Timeout, termination, lock release and requeue.	Self-healing workload recovery.
Credential exposure	CyberArk integration.	No application credentials in bot code.
Ambiguous claims	Preserve work/context and route to Operations.	Controlled human continuation.
6 PM cut-off	Capacity monitoring and end-of-day state transition.	SLA-aware operations.

23. Measurable Outcomes

The outcomes demonstrate why the solution was an enterprise HyperAutomation architecture rather than a narrow bot implementation.

Measure	Outcome
Locations supported	Five operating locations.
Scenario complexity	More than 10,000 rule/scenario combinations considered.
Parallel capacity	Approximately 18 bots in production.
Daily throughput	Approximately 3,000 transactions per day.
Straight-through processing	Approximately 60% deterministic completion.
Human continuation	Approximately 40% routed with context where judgment was required.
Backlog remediation	Two to six month backlog processed/reviewed in approximately one month while BAU continued.
Production quality	Zero recorded production processing defects.
Controlled validation	Two validation incidents; both contained before financial processing.
Infrastructure economics	Terminal Server approach supported multi-session capacity and reduced one-bot-one-VM dependency.

24. Why This Was Hyper-Automation, Not a Bot Project

The program combined requirements engineering, process redesign, SQL/data architecture, RPA execution, CyberArk security, rule externalization, multi-bot orchestration, production infrastructure, runtime monitoring, exception management, human controls, SLA/capacity governance, automated reporting and CoE operating model.

The architecture lesson is clear: successful enterprise automation is not about making a bot execute a task. It is about designing the complete production system around automation.

25. Relevance to Modern Agentic AI

The same production principles now apply to enterprise Agentic AI. Agentic systems still require state management, bounded execution, identity and secrets management, tool authorization, retry and recovery, policy boundaries, observability, human escalation and deterministic operational cut-offs.

Dental Claims provides a strong predecessor pattern for Agentic implementation: the bot worker pool becomes tool execution, SQL state becomes agent/workflow state, configuration-driven rules become policy/guardrail layers, CyberArk remains secrets governance, and human continuation becomes human-in-the-loop approval or escalation.

Dental Claims Pattern to Agentic AI Parallel

Dental Claims Pattern	Modern Agentic AI Equivalent
SQL transaction state / custom queue	Agent workflow state and orchestration.
Configuration-driven rules	Policy and guardrail layer.
CyberArk credential control	Secrets management / workload identity.
Transaction locks	Concurrency and idempotency control.
Timeout and requeue	Bounded execution and retry/recovery.
Ambiguity to manual	Human-in-the-loop escalation.
Preserved work and request state	Context preservation across agents and tools.
Progressive authority model	Observe -> assist -> controlled execute -> bounded autonomy.
Automated reporting	Observability and audit evidence.
CoE operating model	AI enablement and runtime assurance model.

26. Conceptual Claims Data and State Model

The control plane required a persistent state model because the process could not depend on transient bot memory. Every claim record needed a lifecycle from intake to eligibility validation, lock acquisition, processing, save/submit, completion, manual review, requeue or cut-off transition. This state model made the automation explainable and supportable.

A conceptual record included claim identifier, source location, BAU/backlog stream, priority, lock owner, lock time, current status, historical-data readiness, rule set used, processing attempt count, processing start/end time, exception reason, human handoff note and audit timestamps.

- This state model enabled duplicate protection across eighteen workers.
- It allowed backlog and BAU work to be tracked in the same control plane.
- It supported daily reporting, operational escalation and self-healing recovery.

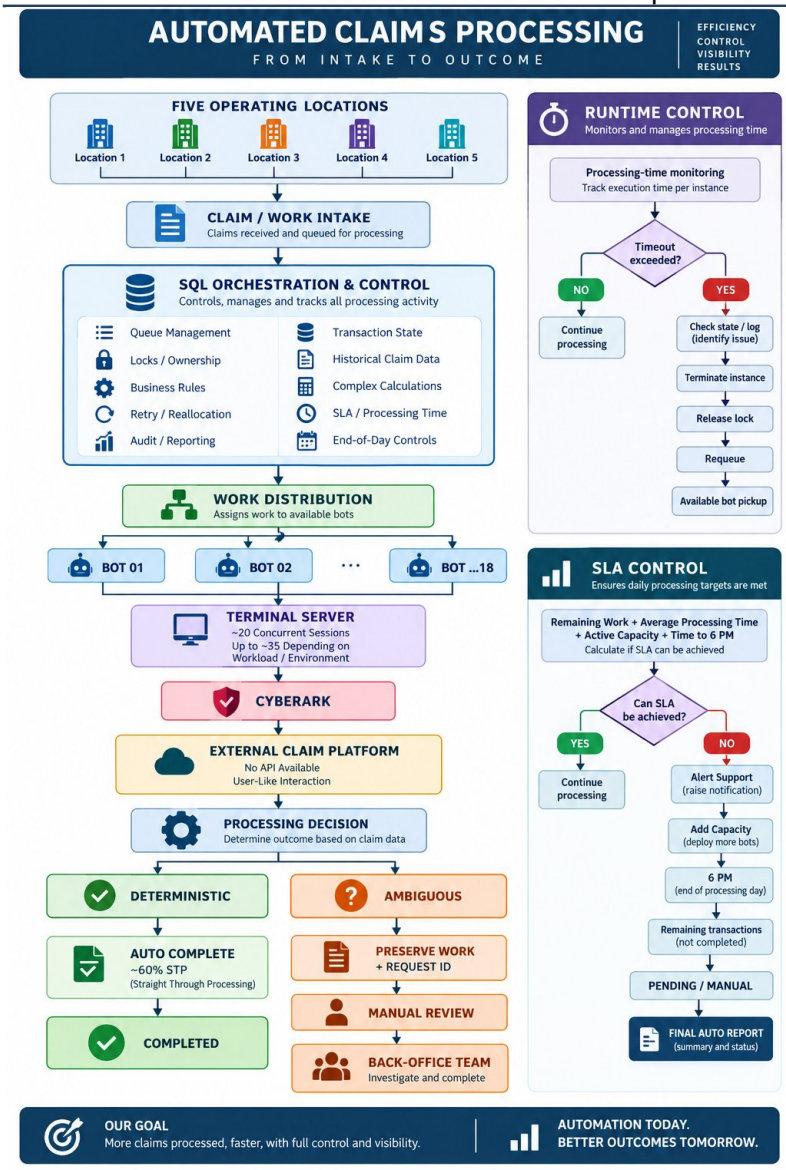
27. SQL Control Plane - Illustrative Table Design

The SQL layer was not only a queue. It operated as the automation control plane. In an enterprise-scale implementation, this layer would be decomposed into transaction, rule, reference, audit, capacity and exception tables. The important design principle is separation of concerns: bots execute work, while SQL controls which work is eligible, owned, blocked or complete.

Illustrative SQL Control Structures

Table / Data Area	Purpose	Example Fields
Claim_Transaction	One row per processable claim work item.	Claim ID, source, stream, priority, status, lock owner, attempt count.
Claim_Rule_Config	Volatile business/government values and rule metadata.	Rule ID, effective date, threshold, deduction percentage, active flag.
Claim_History_Cache	Historical data needed for adjudication and calculations.	Prior claims, provisional conditions, previous processing indicators.
Bot_Worker_Status	Bot/session status and availability.	Bot ID, session ID, heartbeat, current claim, last activity.
Exception_Reason	Controlled reason codes for non-STP handling.	Reason code, description, owner group, SLA.
Audit_Log	Evidence trail for processing, validation and handoff.	Timestamp, transaction ID, action, result, user/bot identity.

28. 30. Architecture at a Glance



29. Bot Worker Lifecycle

Each bot operated as a worker in a governed pool rather than as an independent script. The bot requested eligible work, acquired a lock, retrieved rules and data, executed the live UI steps, recorded results, released the lock and returned to the pool. The bot did not hold permanent ownership of a claim or backlog segment.

This lifecycle allowed the system to add, remove or recover workers without rewriting the business process. It also made support easier because the control plane always knew what each worker was doing.

- Get next eligible transaction.
- Acquire lock and record worker identity.
- Retrieve required rule/reference/history data.
- Execute third-party application processing.

- Record state, exception, completion or handoff reason.
- Release lock and request next transaction.

30. Live Production Validation - Detailed Operating Pattern

Testing in a live money-out system required a carefully bounded operating pattern. The team could not allow uncontrolled test submissions. Reserved claims were isolated, expected outcomes were understood, bot actions were reviewed immediately and the authority to submit was introduced only after evidence was sufficient.

The progressive authority model helped stakeholders accept automation in a high-risk financial process because confidence was earned through evidence, not asserted by development completion.

Progressive Authority Model

Phase	Bot Permission	Operations Control
Observation / dry run	Understand screens, data and decisions.	SMEs validate understanding.
Save Only	Bot processes and saves without final submission.	Operations validates saved work immediately.
Controlled Submission	Bot submits reserved/identified claims only.	Operations validates before financial processing; revoke window retained.
Scaled Production	Bot processes eligible deterministic claims.	Reporting, monitoring, exceptions and support model in place.

31. Backlog Remediation Cadence

The backlog-remediation stream used dedicated capacity and daily control discipline. The objective was not simply to push old claims through faster; it was to process and review them safely while current BAU continued. This required intake visibility, prioritization, rule readiness, capacity tracking and operational review.

- Separate BAU and backlog demand views.
- Daily work allocation based on priority and capacity.
- Monitor throughput and exceptions.
- Route ambiguous claims to Operations with context preserved.
- Produce daily progress and exception reporting.

32. Infrastructure Capacity and Economics

The Terminal Server model changed the economics of scale. Since the claims platform required human-like UI interaction, the team scaled through concurrent sessions rather than unrealistic increases in bot speed. This approach reduced infrastructure duplication and allowed many sessions to run under controlled conditions.

The infrastructure decision supported both throughput and maintainability. It also made it easier to plan capacity around the daily cut-off.

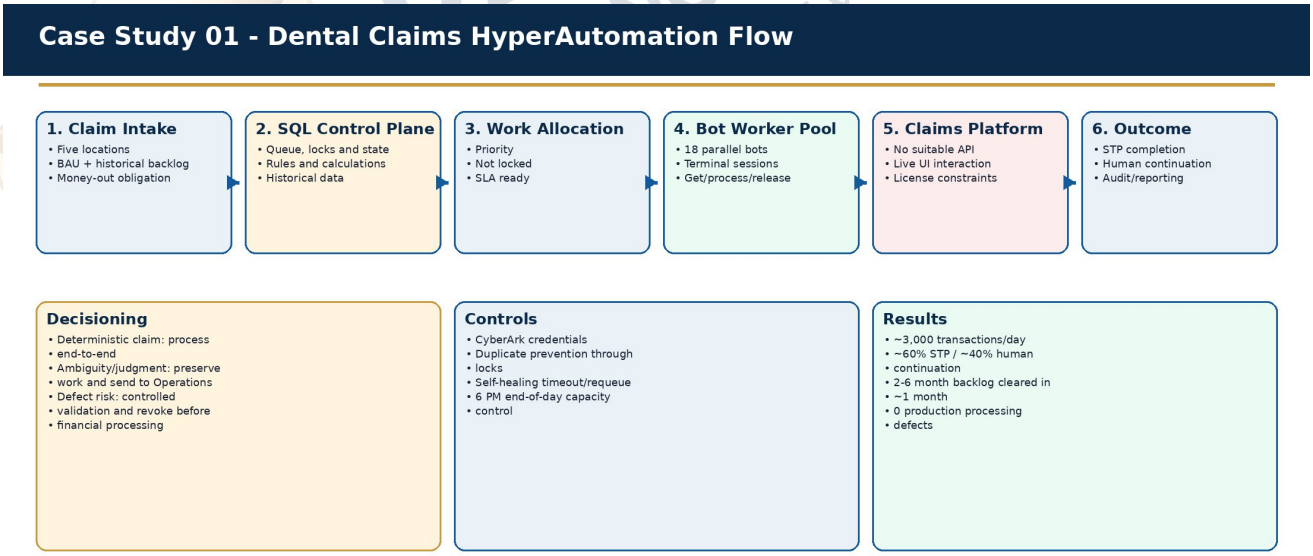
33. Detailed Failure Scenarios and Recovery Controls

The design anticipated failure. A bot could freeze, a terminal session could become unhealthy, an application screen could change, a claim could require judgment, a lock could exceed its expected processing window or capacity could become insufficient before the daily cut-off. Each scenario required a defined operational response.

Failure Scenario Control Matrix

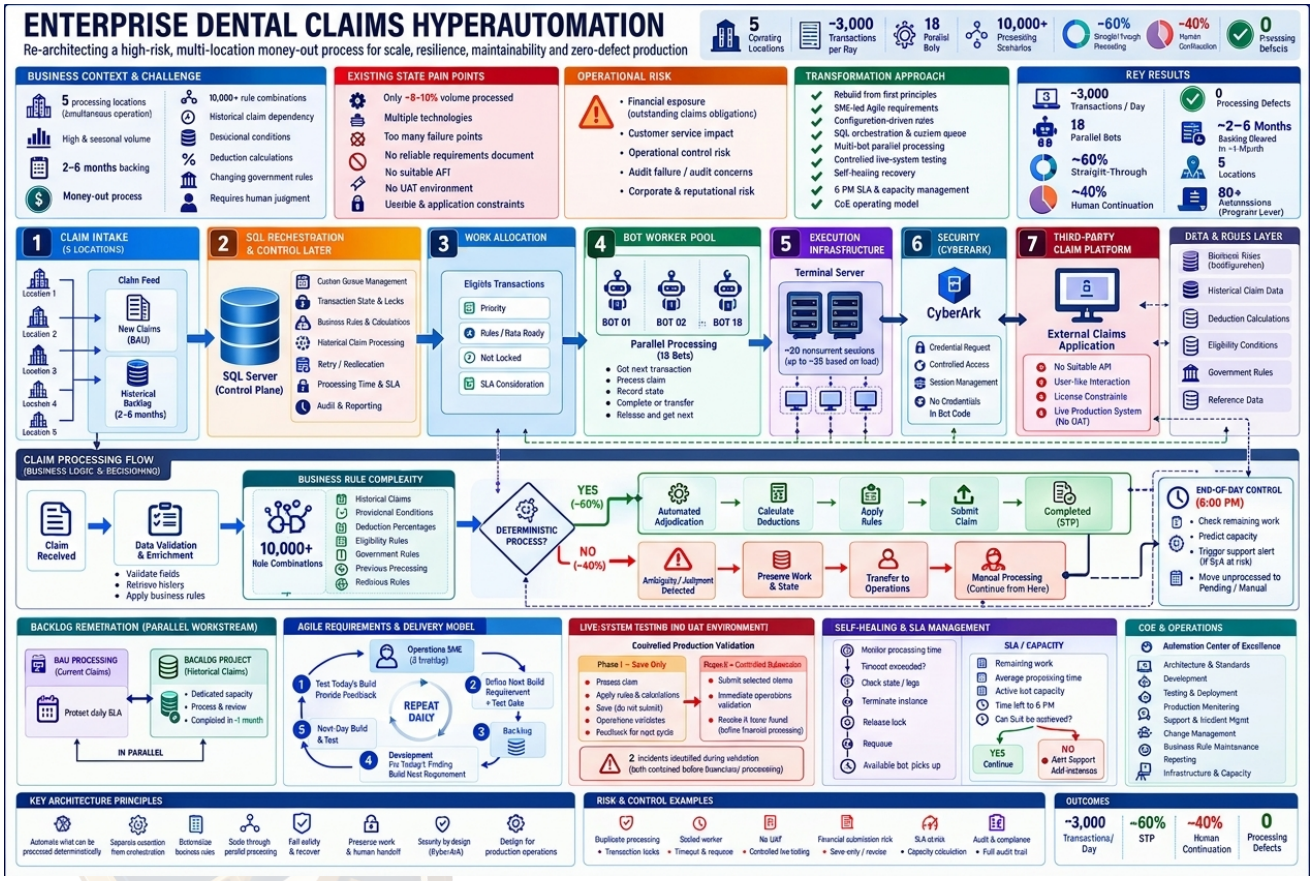
Scenario	Risk	Control
Bot/session stalled	Claim remains locked and unprocessed.	Timeout monitoring, terminate/recover session, release lock, requeue.
Application screen change	Bot failures or wrong processing.	Exception detection, stop affected logic, support review.
Ambiguous claim comments	Incorrect deterministic processing.	Manual review with preserved work/state.
Duplicate acquisition	Two workers process same claim.	SQL locks and transaction ownership.
Cut-off risk	Unfinished claims remain unmanaged.	Capacity projection, alert, pending/manual transition.
Credential issue	Automation cannot access application securely.	CyberArk-managed credential flow and support escalation.

34. 35. End-to-End Flowchart



Proper flowchart: intake, SQL control, allocation, bot execution, UI platform and outcomes.

35. Complete Architecture



Enterprise Dental Claims HyperAutomation