RATHEESH TECHNOLOGY LTD.
AI IMPLEMENTATION & GOVERNANCE

# 2026

## Case Study 02

# ENTERPRISE USER ONBOARDING & ACCESS PROVISIONING AUTOMATION


Author:
**Ratheesh Ramadasan**


*A scalable, resilient and governed multi-application onboarding solution - maximizing operational benefit, not just automation percentage*


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

# 1. Executive Summary

This case study describes a governed enterprise user onboarding and access provisioning automation across a heterogeneous application landscape. New employees or users required access to multiple applications based on unit, role, profile and privilege. The operating challenge was not merely to automate screens; it was to create a scalable, resilient provisioning framework that could select eligible applications, create child transactions, handle application-specific constraints, respect license and bot identity limits, and consolidate outcomes back to the parent request.

Approximately thirty-five applications were assessed. Twenty-two high-value and feasible applications were selected for automation, while others were deferred, partially automated or left manual because of value, risk, feasibility or constraint considerations. The target was not maximum automation percentage; it was maximum operational benefit within technical, licensing, risk and economic boundaries.

# 2. Business Context and Challenge

User onboarding was complex because each request could span multiple enterprise applications, technologies and access rules. Some applications were web-based, some desktop or legacy, and others had mainframe or restricted access patterns. Authentication, licensing, concurrency, availability and security requirements differed application by application.

The manual model produced backlog, inconsistent provisioning, delays, limited auditability and heavy operational effort. A new user might need many application accesses, but one unavailable application should not block all other access provisioning.

- Requests were driven by HR/ITSM intake and included user profile, unit, role and privilege.
- Comments and special instructions often changed how the request needed to be interpreted.
- Applications had different licensing, bot ID and availability constraints.
- High manual effort created backlog and inconsistent onboarding outcomes.
- The solution needed to complete eligible onboarding within a day while preserving control.

# 3. Discovery and Application Assessment

The first step was an application assessment rather than immediate development. The team reviewed application technology, infrastructure needs, authentication model, SSO/MFA requirements, license/concurrency constraints, transaction volume, manual effort, RPA compatibility, operational/security risk and expected business value.

This prevented the program from spending effort on low-value or high-risk automations while ignoring high-impact candidates.

## Application Assessment Outcome

| Category | Count | Rationale |
|---|---|---|
| Automate | 22 | High value, feasible and suitable for controlled automation. |
| Defer / Partial | 6 | Conditional, constrained or requiring partial automation/manual handoff. |
| Remain Manual | 7 | Low value, high risk or not economically justified at the time. |

| | | |
|---|---|---|
| Total Assessed | 35 | Portfolio assessed across technology, risk, licensing and value dimensions. |

# 4. Design Principle - Application-Level Nuance Without Framework Duplication

The core framework was common, but each application had its own nuances. A dedicated application control table allowed application-specific settings, availability, authorized bots, license limits, notes and enable/disable status to be managed centrally.

This was important during application changes. If one application changed or became unavailable, that application could be disabled without stopping the entire onboarding process. Other child transactions could continue.

# 5. Target Architecture Overview

The solution architecture consisted of onboarding request intake, a rules/policy/configuration layer, parent-child transaction creation, smart work allocation, application-specific bots, CyberArk credential management, status/exception management, completion notification and reporting.

The architecture was resilient because parent requests were decomposed into independently executable child transactions. Application-level failure did not necessarily mean parent-level failure.

## Architecture Components

| Component | Purpose | Control Benefit |
|---|---|---|
| Onboarding Intake | Create parent request from HR/ITSM with profile, role, privilege and comments. | Single request context. |
| Rules and Policy Layer | Determine eligible applications and required rules/approvals. | Consistent access decisioning. |
| Parent-Child Transactions | Create child transaction per application. | Independent processing and partial success. |
| Custom Get Transaction | Allocate work only when executable. | Avoids bot idle time and blocked work. |
| Bot Worker Pool | Application-specific automation. | Parallel execution and specialization. |
| CyberArk | Credential retrieval and controlled access. | No credentials in bot code. |
| Status and Exception Layer | Capture success, partial success, unavailable app, license issues and manual needs. | Clear operational visibility. |
| Completion and Notification | Consolidate child status and notify stakeholders. | Transparent request outcome. |

# 6. Parent-Child Transaction Model

A single onboarding request could require access to many applications. The parent request represented the end-to-end business request, while child transactions represented each application-specific provisioning action. This enabled independent processing, retries and exception handling.

For example, if a user required access to twenty-two applications and one application was down, the twenty-one available applications could still be completed. The unavailable application could be held, reason-coded and routed for later processing or manual handling.

# 7. Rules and Policy Layer

The rules layer determined eligible applications based on role, unit, profile, privilege and business conditions. It also checked application-specific prerequisites, licensing constraints, application availability and approval needs.

This layer kept decisioning outside individual bots as much as practical, improving consistency and maintainability.

# 8. Custom Get Transaction - Smart Allocation

A custom Get Transaction concept was created so bots did not simply pick the next item blindly. Before allocation, the framework checked whether the application was enabled, available, valid for processing, within license constraints, mapped to an authorized bot, and within concurrency limits.

This design improved bot utilization and prevented wasted attempts against unavailable or restricted applications.

- Is the application enabled?
- Is the application available now?
- Is the request valid and data-complete?
- Are license and concurrency limits available?
- Is the bot ID authorized for this application?
- Are prerequisite approvals or conditions satisfied?
- If yes, allocate to an eligible bot; if no, hold only that child transaction with clear reason code.

# 9. Application Control and Configuration

A dedicated control table captured application-specific behaviour. This allowed Operations and support to manage application nuances without rewriting the entire bot flow.

The table provided the ability to disable one application during change, outage or defect investigation while preserving the rest of the onboarding pipeline.

## Illustrative Application Control Table

| App ID | Application Type | Enabled | Authorized Bot | License Limit | Availability Window | Notes |
|---|---|---|---|---|---|---|
| APP01 | Claims Portal / web | Yes | Shared Pool | - | 24x7 | Active |
| APP02 | Finance System | Yes | BOT-03 | 1 | 06:00-22:00 | Restricted ID |
| APP03 | HR Legacy | No | BOT-05 | 1 | 24x7 | Under change |
| APP04 | Analytics | Yes | BOT-04/05 | 2 | Business Hours | Limited concurrency |
| APPxx | Other apps | Configurable | Mapped as required | Configurable | Configurable | Application-specific nuance captured |

## 10. Bot Worker Pool and Application-Specific Automation

Bots were designed around application-specific patterns where needed. They executed provisioning tasks, recorded result/state, and released themselves to acquire the next eligible item. The framework maximized reuse without ignoring application differences.

Application-specific bots allowed targeted testing, support and change impact management. When one application changed, the impact could be contained to that application's bot/configuration rather than destabilizing the entire onboarding solution.

## 11. Licensing, Bot Identity and Concurrency Governance

Licensing and bot identity constraints were first-class design considerations. The framework checked license limits and authorized bot IDs before allocating work. This avoided failed logins, license contention and unauthorized access attempts.

The same idea becomes very relevant to Agentic AI: a tool or bot should not inherit broad access simply because it is part of an automation platform. Access must be scoped to the specific business function.

## 12. Authentication, Security and CyberArk

Application credentials were not hard-coded. They were retrieved through CyberArk with controlled access, session governance and auditability. This ensured the automation did not create unmanaged privileged identities.

CyberArk also supported operational control because credentials, sessions and access boundaries could be managed outside bot code.

## 13. Application Availability and Downtime Handling

Application downtime and frequent changes were significant risks. The architecture handled this by checking availability before allocation and allowing applications to be enabled or disabled independently.

If an application was unavailable, the related child transaction could be held while the rest of the parent request continued. This avoided all-or-nothing failure.

## 14. Unstructured Comments and Special Instructions

Many onboarding requests contain comments, notes or special instructions that influence processing. In the classic automation model, such comments often required manual interpretation. The framework captured reason codes and provided paths for manual handling where the request was not deterministic.

In a modern Agentic extension, comments can be diagnosed by an agent to extract intent, identify required applications, detect ambiguity and improve downstream routing while still preserving deterministic controls for access and authorization.

## 15. Partial Completion and Exception Management

The solution was designed to support success, partial success, application unavailable, license not available, business exception, system exception and manual intervention. A child transaction could fail or defer without collapsing the entire parent onboarding request.

Clear reason codes allowed Operations to understand what remained and why.

### Exception and Status Model

| Status | Meaning | Operational Handling |
|---|---|---|
| Success | Application access provisioned. | Update child and parent progress. |
| Partial Success | Some applications completed; others pending or failed. | Notify with outstanding items. |
| Application Unavailable | Target application down or disabled. | Hold/retry or manual queue. |
| License Not Available | Concurrency/license limit reached. | Requeue when available. |
| Business Exception | Rule/prerequisite/approval issue. | Manual review or approval. |
| System Exception | Technical failure. | Support investigation and retry. |
| Manual Required | Non-deterministic comment or unsupported condition. | Send to Operations with context. |

## 16. Completion and Notification

The completion layer consolidated child status back to the parent request. Stakeholders such as HR, IT and the user could be notified of completed access and unresolved items.

This created visibility and avoided the common problem where requestors know that onboarding is “in progress” but cannot see which applications are complete and which are blocked.

## 17. End-of-Day Control and Daily Reporting

The operating model included end-of-day controls to check remaining work, monitor capacity, trigger SLA-risk alerts, move unprocessed transactions to pending/manual states and generate daily reports.

This made onboarding a managed production service rather than a batch of unattended scripts.

## 18. Agile/Kanban Delivery Model

Delivery was structured around discovery, feasibility, design, development, unit testing, business validation, integration and controlled production rollout. Applications could be developed and tested in parallel where practical.

Independent testing per application reduced risk and enabled controlled rollout across the portfolio.

## 19. Key Outcomes

The solution significantly reduced onboarding backlog and enabled eligible requests to be delivered within a day. It improved bot utilization, licensing compliance, operational visibility, auditability and resilience to application changes.

| Measure | Outcome |
|---|---|
| Applications assessed | 35 |
| Applications automated | 22 high-value applications |
| Turnaround | Within a day for eligible onboarding requests |
| Backlog | Significant reduction |
| Manual handling | Only unresolved application-level items routed to Operations |
| Control | Improved auditability, reason codes and reporting |

| | |
|---|---|
| Resilience | Application-level disable/hold prevented one change from stopping the whole solution |

# 20. Agentic AI Evolution

This onboarding framework naturally evolves into Agentic AI. An agent can interpret comments and special instructions, classify ambiguity, retrieve SOPs, determine the applications likely required and create structured child transactions. However, deterministic controls must still govern authorization, application eligibility, bot identity, licensing and execution.

The modern version would use an MCP/configuration layer to expose approved tools, while individual agents receive only the access required for their function.

# 21. Detailed Application Discovery Template

The application discovery step produced the knowledge needed to decide whether to automate, defer, partially automate or leave manual. This prevented a one-size-fits-all approach across thirty-five applications with different technology and control profiles.

## Discovery Fields

| Dimension | Questions Answered |
|---|---|
| Business value | How much manual effort/backlog does the application represent? |
| Technology | Web, desktop, legacy, mainframe or mixed? |
| Authentication | SSO, MFA, service account, restricted ID or manual token? |
| Licensing | How many sessions/users can operate concurrently? |
| Availability | 24x7, business hours, maintenance window or unreliable? |
| Security risk | What privilege does the access grant? |
| RPA suitability | Stable UI/API? predictable forms? exception complexity? |
| Support effort | How often does the application change? |

# 22. Access Rule and Role/Profile Mapping

Provisioning logic depended on the relationship between user role, unit, profile and privilege. The framework needed to understand which applications were eligible for a request and which required approval or special handling. This rule logic was centralized so application bots did not independently reinterpret access policy.

# 23. Custom Get Transaction Algorithm - Detailed Logic

The custom allocation logic was the heart of the framework. It ensured that bots were assigned work only when the target application and required prerequisites were ready. This avoided unnecessary failures and improved throughput.

## Allocation Checks

| Check | Reason |
|---|---|
| Parent request active | Avoid processing cancelled/completed onboarding. |

| | |
|---|---|
| Child transaction pending | Select only outstanding app-level work. |
| Application enabled | Allow project/support to disable an app under change. |
| Application available | Avoid processing during downtime/maintenance. |
| License available | Prevent login/license contention. |
| Authorized bot mapped | Ensure only permitted bot identity accesses that app. |
| Required approvals complete | Avoid unauthorized provisioning. |
| Concurrency available | Respect application/session limits. |
| No duplicate lock | Ensure one worker owns the child transaction. |

# 24. Handling Application Change Without Platform-Wide Outage

One of the strongest design points is application-level isolation. If APP03 was under change, it could be disabled in configuration. The rest of the onboarding workflow continued for other applications. This avoided a common automation failure pattern where one UI change brings down an entire bot estate.

# 25. Detailed Status Consolidation Logic

The parent request status was not a simple completed/failed flag. It was derived from child transaction status. This allowed transparent reporting such as completed, partially completed, pending, manually routed or failed with reason codes.

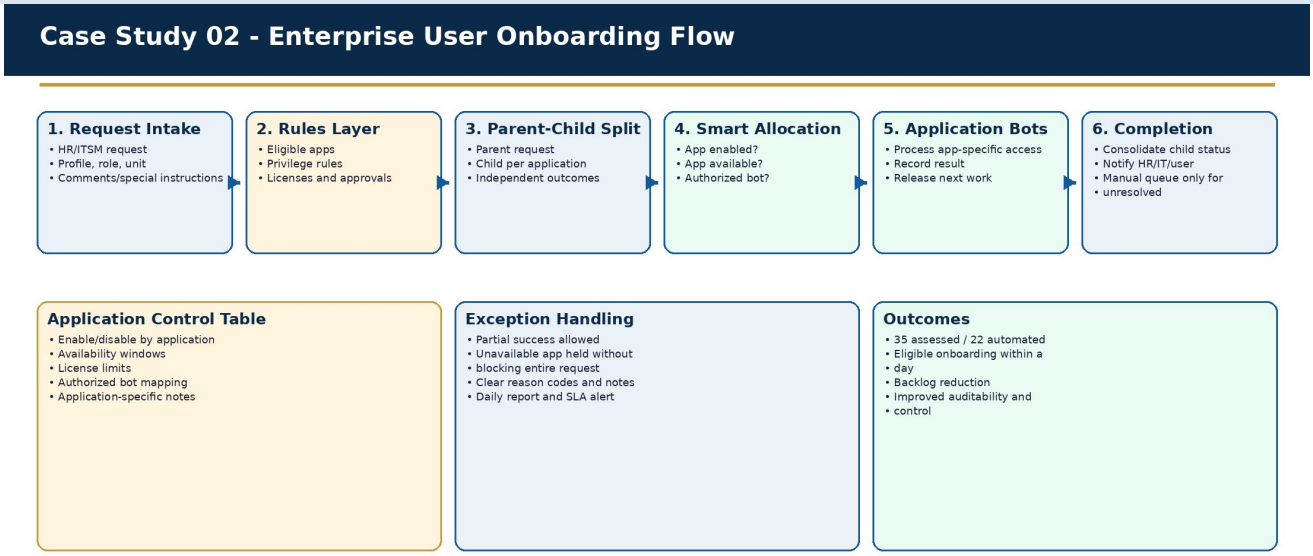
## Parent Status Derivation

| Child Outcomes | Parent Status | Notification Message |
|---|---|---|
| All completed | Completed | All eligible application access has been provisioned. |
| Some completed, some pending | Partially Completed | Access completed for available applications; remaining items pending. |
| Application disabled/unavailable | Pending / Awaiting Application | Specific application is under change or unavailable. |
| Approval required | Pending Approval | Access request awaits required approval. |
| Manual condition | Manual Review | Special instruction or ambiguity requires Operations review. |
| System issue | Support Required | Technical issue assigned with reason code. |

# 26. Agentic Extension for Comments and Special Instructions

In the Agentic evolution, free-text comments can be interpreted to identify access intent, detect special conditions and classify ambiguity. However, the agent should not directly provision access based on comments alone. It should convert unstructured comments into structured candidate actions, then deterministic policy and authorization controls should decide eligibility.

# 27. End-to-End Flowchart




*Proper flowchart: parent-child onboarding transaction model with application-level control.*

# 28. Complete Architecture