EFTEDRA

ARCHITECTURE WHITEPAPER

Claude Inside IBM BAW

An architecture for embedding a model assistant into Business Automation Workflow processes and screens — without putting the model in the run path, and without exposing enterprise data to it.

EFTEDRA — Smart Workflow Solutions
Version 1.0 · 2026
eftedra.com

Contents

1. Why the assistant belongs inside the process

> Most enterprise AI pilots put the model beside the work. The person doing the job leaves the system of record, pastes context into a chat window, gets an answer, and carries it back by hand. The pilot demonstrates capability and changes nothing about cycle time, because the expensive part was never the thinking — it was the switching, the re-keying and the absence of any audit trail.

A workflow engine already knows things a chat window cannot: which case this is, which step it is on, who is assigned, what data has been gathered, and what must happen next. An assistant that runs *inside* the process inherits all of it. It does not need to be told the context because the context is the task.

It also inherits the governance. A task in IBM Business Automation Workflow is assigned, claimed, timed, escalated and audited. If the assistant's contribution is a step in that process rather than a conversation next to it, then every suggestion it made, every document it classified and every value it extracted lands in the same audit trail as everything else — attributable, timestamped and reviewable.

The three questions that decide the architecture

Every serious enterprise deployment eventually reduces to three questions, and it is worth answering them before writing any code:

- **What can the model see?** Not in principle — in bytes. Which fields, which documents, which rows.
- **What can the model do?** Which actions are reachable, and what is the worst outcome if it chooses wrongly or is manipulated into choosing wrongly.
- **What happens when it is wrong?** Who notices, how quickly, and what does the audit record show six months later.

The architecture in this paper is organised entirely around making those three answers short, specific and demonstrable.

2. What already exists, and what is missing

IBM has moved in this direction. Business Automation Workflow now ships an assistant experience in Workplace, powered by watsonx.ai, which helps users find, filter and summarise tasks, and generate content such as emails and descriptions. Recent releases lean on enterprise foundation models — the Granite family, Llama, Mixtral — for content generation inside business-critical automations.

That establishes the pattern is legitimate. It leaves three gaps that matter to organisations with specific processes rather than generic ones:

Gap	Consequence
Generic, not process-specific	A platform assistant helps with tasks in general. It does not know your claims triage rules, your document taxonomy or your approval thresholds unless something teaches it, per process.
Model choice is the platform's	Organisations that have evaluated models and standardised on one — for capability, for commercial terms, or for a data-processing agreement their legal team has already cleared — cannot simply substitute it.
Assistance, not configuration	Summarising a task is useful. Letting a business user safely configure how the application behaves — a new report, a new view — is a different and larger problem, and it is the one that actually removes work from the delivery backlog.

The approach described here is complementary rather than competitive: process-specific assistant components, built on Claude, that a process designer installs into a particular flow or screen, with the configuration and safety boundary defined per use case.

3. Two integration surfaces: assistant tasks and coach views

There are exactly two places an assistant needs to live inside BAW, and they solve different problems.

3.1 The assistant task

An assistant task is a step in the process with no human in it. The token arrives, the task runs, it writes its output into the run data, and the token moves on. It is modelled as an ordinary automated activity, which means it participates in the flow exactly like any service task — it can be routed around, retried, or skipped by a decision.

Each assistant task is configured with three things:

- A **system prompt** defining its narrow job in this process.
- A **tool set** — the specific data-manipulation operations it may call, each with a JSON Schema.
- An **output contract**: the schema of what it must return and the run-data keys those values are written into.

Typical uses are classification with vision (what kind of document is this attachment), extraction (pull these fields out of it), and normalisation (map this free text onto our controlled vocabulary).

3.2 The coach view

A coach view is the assistant appearing inside a screen a person is already using. The case worker does not leave BAW; a panel in their task interface offers the model's reading of the case, drafts a response, or explains what the next step needs.

Coach views matter because they keep the human in the loop *by construction*. The model proposes; the person disposes; the process records what was accepted. For any judgement that carries consequence, this is the right shape — and it is why an architecture that only offered headless assistant tasks would be incomplete.

> **On external implementations**
>
> Where the interface should live outside BAW entirely — a modern web application owning the user experience — BAW's *external implementation* mechanism keeps the task in the engine (assignment, teams, escalation, audit) while your application renders it. Coach view and external implementation are not competing choices; they answer different questions about where the interface belongs. Use a coach view when the work should stay in Workplace, an external implementation when it should not.

4. The safety model

Four rules do the work. Each is structural rather than advisory, because principles erode and structures do not.

4.1 The model never routes

The assistant produces *data*. It classifies, extracts, drafts and suggests. It never decides where the case goes next. Routing is evaluated by a deterministic condition language over the run data, against a published, immutable version of the flow.

This is what makes the system explainable after the fact. When someone asks why a case went to enhanced review, the answer is a rule and a value, both recoverable from the audit record — not an appeal to how a model weighed things on a particular afternoon.

4.2 The model only acts through a tool surface

Every capability is a named tool with a JSON-schema'd payload. There is no general escape hatch, no raw query interface, no arbitrary code execution. The blast radius is therefore not a matter of how well the prompt was written — it is the union of what the declared tools can do, which is a fixed, reviewable list.

4.3 The model sees the least it can

Context is assembled per task from the specific fields that task needs. This is deliberate and it is the rule most often skipped, because passing the whole case object is easier. It is also where most real data-exposure incidents originate.

4.4 Every output is validated before it is trusted

Schema conformance is necessary and insufficient. Business rules are checked separately and deterministically: totals reconcile, dates fall in range, references resolve against master data, enumerations are members of the set. A confidence signal and a verbatim `source_text` span for each extracted value make human review fast and give you an automated check — if the quoted span is not present in the source document, something is wrong, and no human is needed to notice.

5. Worked example: document intake

A claim arrives with attachments of unknown type and quality.

1. **Receive** — the process starts, attachments land in run data.
2. **Classify (assistant task)** — Claude, with vision, labels each attachment against a closed taxonomy expressed as an enum. Anything it cannot place is `unknown`, which is a valid answer rather than a guess.
3. **Extract (assistant task)** — per document type, a schema'd extraction with `source_text` and confidence for every field.
4. **Validate (deterministic)** — ordinary code. Do the line items sum? Does the policy number exist? Is the date plausible?
5. **Decision (deterministic)** — low confidence or failed validation routes to a human review task; otherwise straight through.
6. **Review (coach view)** — the reviewer sees extracted values beside the quoted source spans and corrects what is wrong.

Two properties are worth naming. The model appears twice and routes nothing. And step 6 is where correction data accumulates — the record of what humans changed is the most valuable evaluation asset the system will ever produce.

6. Worked example: safe self-service reporting

The harder and more valuable case. A business-unit manager wants a report page showing instance progression alongside real application data. Today that is a ticket, a developer, and a release.

The naive implementation — give a model database access and let it write queries — is the pattern the industry spent 2026 learning not to deploy. The failure modes are well documented: a query that is syntactically valid and still returns data the requester should never see; the confused-deputy problem, where the agent's elevated privileges silently bypass the user's own permissions; and generated statements that are destructive or that dump entire tables. Anthropic's own published guidance on securing text-to-SQL treats these as the default risks to design against, not edge cases.

The architecture here avoids them by never granting the capability in the first place.

Element	Rule
Context	The model receives a *description* of the data structures — table and column names, types, relationships, semantics. It never receives rows. Not sampled, not anonymised: none.
Surface	Only pre-approved, sanitised views are described. Columns that must not leave the system are absent from the described surface, so they cannot be selected by a query that does not know they exist.
Output	A JSON configuration document conforming to a published schema, plus read-only queries scoped to those views. The configuration — not free-form code — is what drives the application.
Credentials	Execution uses a read-only principal with access limited to the same views. The model's output cannot exceed that grant regardless of what it emits.
Validation	Generated queries are parsed and checked before execution: read-only, known views only, row limits enforced, no unbounded joins.
Review	The manager sees the report before it is saved. Publishing is an explicit human action, recorded.

The result is a business user configuring their own reporting without a developer, and without any path by which proprietary data or restricted columns reach the model. The safety property does not depend on the prompt, or on the model behaving well. It follows from what was never made reachable.

Why config-as-output is the important idea

Asking a model for *configuration* rather than *code* collapses the risk surface. Configuration is schema-validated, diffable, reversible and inspectable by someone who cannot read code. It cannot contain arbitrary logic because the schema does not admit any. Everything the configuration can express was designed in advance by an engineer; the model is only choosing among sanctioned options.

7. Operations, cost and failure behaviour

Failure is a design decision

Assistant tasks fail like any integration — timeout, rate limit, outage — and each one needs a declared behaviour: retry with backoff, route to a human, or proceed with a flag. The unacceptable option is an unhandled exception that leaves an instance parked where nobody is looking.

Degrade honestly

Every assistant step should have a defined behaviour when the model is unreachable, and in a well-designed process that behaviour is almost always *route to a human*. The process continues at reduced throughput rather than stopping. This also answers the procurement question about third-party dependency without requiring a promise about uptime.

Cost is a per-step property

Because assistance is a step rather than an ambient capability, its cost is attributable per case, and model choice is a per-step decision. Classification against a closed taxonomy does not need the same model as drafting a customer response; a smaller, faster model at the front of a pipeline can pay for a more capable one at the point of judgement.

Evaluate on your own corrections

Public benchmarks will not tell you whether extraction works on your documents. The review step in section 5 produces a growing, labelled set of exactly the cases your process sees. That corpus — not a benchmark — is what should gate any change to a prompt, a schema or a model version.

8. An evaluation checklist

Whether you build this yourself, buy it, or ask us, these are the questions worth putting to any assistant-in-workflow proposal:

- Can you show, field by field, exactly what the model receives on a given step?
- Is the list of actions the model can take finite, declared and reviewable?
- Does the model influence routing? If yes, how is a routing decision explained a year later?
- What is executed in production — a live definition, or an immutable published version?
- What happens to an in-flight case when the model API is unavailable?
- For any generated query: is the credential read-only, is the surface a restricted view, and is the statement validated before execution?
- Where do human corrections go, and are they used to evaluate changes?

- Can the model be changed without changing the process?

An architecture that answers all eight is not necessarily the right one for your organisation. But one that cannot answer them is not ready for a regulated process, however well it demonstrates.

About EFTEDRA. We build workflow automation on IBM Business Automation Workflow and Claude — assistant tasks and coach views designed to install into processes our clients already run. This paper describes the architecture we build to. Components described here are in active development; working demonstrations are available on request.