PROFORMA GLOBAL RESEARCH

Agentic AI: Navigating and Executing Through Semantic Layers Based on the Organization's Position on the AI Maturity Curve

Matt Rollings, Founder and Principal, Proforma Global

Whitepaper · 2026-05-18

Executive Summary

An agent operating in an enterprise financial system performs two categorically distinct kinds of work: navigating to the information and operations the question requires, and executing against what it finds. The difficulty curves for these two modes diverge as the organization scales. Navigation grows easier as the semantic layer matures and absorbs more of the resolution work. Execution grows harder as each new business process, functional area, or use case introduces another semantic frame the agent must reason within. The architectural investments that support each mode must respond to that divergence, and they admit different right answers at different points on the organization's AI maturity curve.

The dominant failure across enterprise AI programs is the pursuit of a universal architectural answer. Reference architectures from vendors, conference circuits, and consulting practices describe a single correct approach. The approach is correct for one stage of maturity and damaging at others. An organization that adopts a late-stage architecture early over-engineers itself into delivery paralysis. An organization that holds an early-stage architecture too long discovers it cannot accommodate the plurality of the work it has accrued.

This paper sets out the two modes of agentic work, the five stages of organizational AI maturity in financial systems, the relationship between maturity and architecture, and why the discipline of matching the architecture to the actual stage separates working programs from showcase ones.

1. Two Modes: Navigation and Execution

An agent in an enterprise financial system divides its time between two operations whose architectural requirements diverge sharply.

Navigation is the resolution of ambiguous input to definite reference. The agent must identify which ledger contains the data, which account the user means, which business process governs the request, which dimension intersection holds the answer, which rule applies to the calculation. The system of record is not laid out for human or model intuition. Navigation is the work of producing a definite address from an indefinite question.

Execution is the application of logic to a resolved reference. Posting a journal entry. Validating an accrual. Recomputing a forecast. Generating a report. Deploying an artifact. The operation must apply the correct rules, accommodate the correct exceptions, satisfy the correct validations, and produce an output that the downstream consumer can accept. Execution is the work of doing, not finding.

The two modes appear similar from the outside and are not the same problem. Navigation responds to investment in the semantic layer between the agent and the data. As the layer matures, it absorbs the resolution work, exposes controlled vocabulary, encodes structural context, and answers most navigation queries before the model is engaged. Execution responds to investment differently. The rules that govern an operation change with the business process, the functional area, the use case, the entity, and sometimes the specific transaction, and no general layer captures the variation cleanly.

The divergence has an architectural consequence. The two modes deserve different investments at different points on the curve, and the program that invests evenly across them at every stage is misallocating against the work it is actually doing.

---

2. The AI Maturity Curve in Financial Systems

Five stages characterize most enterprise AI programs in financial systems. The stages are not chronological in calendar time. Organizations move at different rates, skip stages, regress under leadership change, and sometimes operate at different stages in different functional areas concurrently. The stages describe architectural reality, not project schedule.

Stage 1. Exploration

This is the proof of concept stage. The agent performs one operation for one workflow with one set of users. The semantic layer is whatever the team produced in the first month. Data access is whatever the existing system exposes. The success criterion is that the demonstration runs. Most organizations remain here for three to nine months, longer if the vision changes.

The correct architecture at this stage is the smallest one that delivers the use case credibly. A bounded state machine, a handful of deterministic tools, a thin semantic layer over the existing data model. Investment in governance, multi-context layers, or universal frameworks is wasted

because the organization does not yet know what it is building toward. Premature architecture is the recurring failure: the demonstration takes too long to release, leadership loses faith, and the program is cancelled before it earns the right to advance.

Stage 2. First Production

One or two agents have left the laboratory. Real users are using them. Real data is flowing. Reliability, observability, and guardrails are now the constraints on progress. A thin semantic layer crystallizes around the systems the agents touch most directly, codifying the resolution patterns the team had to encode the hard way during exploration.

The correct architecture at this stage supports operational reliability for the agents in production. The semantic layer captures what exploration taught. Execution logic remains largely hand-written for the use cases at hand. Cross-domain concerns are visible without yet being pressing.

The trap at this stage is the second use case. The team wants to demonstrate the pattern generalizes and pushes a second agent into production before the first has matured. The result is two unreliable agents instead of one reliable one. Stage two is a discipline of restraint: finish productionizing what is in place before opening the next workstream.

Stage 3. Scaled Deployment

Multiple agents operate across multiple functional areas. The team discovers that the same concept means different things in different contexts, and the contexts now extend past data definitions into processes and activities. "Close" is a sequenced set of reconciliations in accounting and a forecast freeze in FP&A. "Approval" carries different thresholds and rules in AP than in journal entry posting. "Variance" lives in accounting, planning, and statistical contexts with different math and different ownership in each. A traditional data dictionary handled the naming layer of this problem. With the emergence of agents, the dictionary must cover data, processes, activities, and the cross-discipline meaning that language models would otherwise conflate. The instinct, faced with that expansion, is to dissolve the contexts into a single unified model that satisfies all of them. The instinct surfaces here and is usually indulged. Most enterprise programs occupy this stage today.

The correct architecture admits bounded contexts as a first-class design problem. The semantic layer fragments into multiple bounded layers, each authoritative within its scope. Execution logic for each business process receives its own treatment. The work is to make the contexts explicit and enforce the boundaries between them.

Attempting to dissolve the boundaries in favor of a unified construct is the fatal mistake of this stage and the most common one. The contexts being unified are genuinely different, the unified model never settles, every team negotiates semantic compromises, and every release waits for a model that cannot stop moving.

Stage 4. Enterprise Integration

Agents, use cases, functional areas, and overlapping semantic layers begin to contradict each other as each action becomes nuanced and conditional. Orchestration and conditional execution against N parameters and approved use cases become intractable.

At this stage solutions diverge from simple process flows that can be navigated cleanly into highly nuanced hierarchies where business logic is inherited across five or six different semantic and logic layers.

Almost all organizations have experienced the failure at this stage, where process and technology collide. Process standardization is what enables technology to assist, and standardization in practice means quietly agreeing to ignore certain edge cases. The risk runs in both directions. Pushed too far, normalization introduces rigidity the business cannot operate within. The opposite failure drives the work into spreadsheets and tribal knowledge, where it becomes operationally ungoverned.

The architecture that succeeded at stage three produces operator overhead at stage four. Operators discover that the contradictions between contexts are being maintained outside the system: in spreadsheets, in tribal knowledge, in the heads of senior controllers who have been there long enough to remember which framework applies in which case.

The correct architecture captures cross-context policy explicitly. Inheritance precedence between frameworks. Conditional rules per business process. Validation cascades that activate selectively. The work has shifted from constructing layers to governing them.

Most organizations will remain in a perpetual cycle of trying to master their processes, data, and systems at this stage.

Stage 5. Mature Multi-Domain Operation

The plurality of the financial domain is the central design problem. The agent navigates several models that overlap and contradict. Execution requires the agent or the architecture to know which frame applies to which question. The system resembles a federation of bounded models

with explicit translation between them. Policy lives in tables. Conditional logic lives in evaluatable rules. The agent's reasoning loop is stable; what changes per query is the layer it traverses and the rules that govern execution within it.

Almost no organization is here today. The published reference architectures describing stage-five operation are aspirational. The vendors marketing stage-five frameworks to stage-two organizations are doing real damage.

Oracle EPM Cloud, as a representative enterprise financial platform, illustrates the curve. A stage-one program builds an agent that completes a single form. A stage-three program builds agents across forms, reports, navigation flows, and integrations and discovers that the same dimension carries different semantics in different cubes. A stage-five program operates across consolidation, planning, and statutory reporting and accepts that no single semantic model spans all three coherently.

---

3. The Graph Question, Across the Curve

The state machine graph as the architectural foundation for agentic execution is a useful case study because the correct answer changes at every stage. The simple answer cleanly selects it as the solution. The advanced answer determines best uses and variants by use case, complexity, and workload. The expert answer dismisses the question as malformed without a maturity context. All three are correct, at different stages.

At stage one, a state machine graph is the right answer without qualification. The agent has one job. The graph encodes the steps. The framework supplies replayability, audit, and resumability without custom engineering. The pattern accelerates the pilot.

At stage two, the answer holds. The graph from the pilot scales to a small population of production agents. The semantic layer beneath absorbs the resolution work the graph does not address. The team accretes tools, refines transitions, and releases the next use case on the same pattern.

At stage three, the answer fractures. The graph that handled one functional area must extend to several. The team either grows the graph until conditional branching dominates every node, in which case it becomes unmaintainable, or it splits into one graph per use case. The second move is sound. The engine still provides the operational value, and this is how workflow engines

run in every large bank today. The wrong move at stage three is the unified graph attempting to serve every use case. The right move is many bounded graphs, each owned by the context it serves.

At stage four, the limit of the graph as the architectural foundation becomes visible. The plurality of the work outpaces the structural commitment the graph requires. A graph encodes one set of relationships, one set of branching rules, one set of inheritance paths. Financial work at this stage routinely demands several at once, and the selection between them is a judgment that lives outside any individual graph. Teams find themselves maintaining the cross-graph judgment in places the system cannot see.

The reality is that as an organization scales out its agents and use cases, a fundamental question arises. How do we scale intense complexity and optionality? The available answers are three:

A) Simplify the processes, ignore exceptions, normalize the data and logic.

B) Accept optionality as a feature and implement parallel solutions for different business lines requiring specificity.

C) Attempt to construct a more elegant unified solution that can traverse the complexity coherently.

At stage five, a mosaic of designs and solutions is deployed, with balance achieved through technical elegance adapted to individual business processes rather than imposed across them. The discipline that produced success through stages one to four is simplicity, normalization, and restraint. Stage five operates by the inversion.

Over-reliance on language models to interpret highly complex financial data reliably, without hallucination or silent failures masquerading as successes, is a catastrophic design risk in this domain. The number of domains and semantic layers an LLM would need to traverse to fully understand the optionality and the exact operations exceeds what current models can reliably hold. Organizations at stages four and five will spend years, if not decades, attempting to construct a working universal traversal and execution model.

---

4. What Organizations Get Wrong

Three failures recur across enterprise AI programs in financial systems. Each is the consequence of mismatching the architecture to the actual maturity.

The first is over-engineering. A stage-two organization adopts a stage-five reference architecture on the recommendation of a vendor or a consultant. The team spends two years constructing bounded contexts, policy tables, federation layers, and inheritance evaluators against use cases it has not yet encountered. Delivery slips. The pilot is never released. Leadership concludes the program is failing and either replaces it with a simpler one or cancels it outright. The architecture was correct for a future the organization had not earned the right to design for.

The second is under-engineering. A stage-four organization tries to scale a stage-two architecture. The team that built the original semantic layer has moved on. Their successors continue adding agents to the same architecture, encoding new cross-context contradictions in code rather than in policy, hand-resolving conflicts that should have been governed structurally. The architecture accumulates technical debt the program cannot retire without halting delivery. The architecture was correct for an earlier stage that the organization has outgrown without acknowledging it.

The third is mis-reading the stage. The organization believes it occupies stage two when it has actually reached stage three, or believes it occupies stage four when it remains at stage two. The diagnostic is not the headcount of agents in production. The diagnostic questions are: how many functional areas does the agent population span; how many business processes does each agent participate in; how many contexts does a single concept have to mean different things in; how many of those contradictions are being maintained outside the system. The answers reveal the stage. The headcount, the org chart, and the platform investment do not.

The pattern across all three failures is identical. Architectural decisions are made against an imagined organization rather than the one in front of the team. The imagined organization is usually either the one the vendor's reference architecture presupposes or the one the team aspires to be in five years. Neither is the real one. The real one occupies a specific stage with specific work in front of it, and the architecture either matches or fails to.

---

5. The Discipline

Building agentic AI for financial systems is a discipline of stage-matching. The program that asks what work the organization currently performs and constructs the architecture that supports that work, while preserving the option to advance to the next stage, outperforms the program that pursues the most sophisticated architecture available. The discipline also assumes architects who hold business process knowledge, agent design, technical depth, and

financial systems experience in a single head. Programs that distribute these across siloed specialists discover that the integration that matters happens in the architect's head and nowhere else, and that those architects are scarce.

The natural objection is one of future-proofing: every CIO wants an architecture they will not have to rebuild. The objection misreads the work. Future-proofing for stage five from stage two produces a stage-two architecture wearing stage-five framing: expensive to build and incapable of delivering at either stage. The architectures that mature successfully are designed for the work in front of the team and architected to advance, with each stage's investments preserved as the next stage's foundation.

At stage one, the work is to keep the architecture small. A bounded set of tools, a single state machine, a thin semantic layer over the existing data model. The pilot must be delivered.

Stage two requires repeatability. Agents in production must produce the same answer to the same question every time, and the semantic layer codifies what exploration taught.

Adding the next agent at stage three must not destabilize the existing ones. Bounded contexts give the program a path to growth without merging everything into a single architecture no team can govern.

By stage four, governance is the binding constraint. The architecture must capture cross-context policy explicitly. Contradictions that earlier stages tolerated in code or tribal knowledge become policy artifacts the organization can review, version, and own.

Stage five operates as federated coherence. Multiple bounded models, explicit translation, condition-evaluated execution, and stable reasoning loops above all of it.

The objective at every stage is the same in character and different in form. As organizations implement AI and the data architectures that support it, they should strive for simplicity, repeatability, and stability, not purist technical elegance. The purist's reference architecture is correct for a stage no specific organization currently occupies, and the program that pursues it for its own sake constructs an architecture that satisfies no actual workload.

This counsel is calibrated to the population the paper addresses, which is most organizations operating between stages one and four. The rare organization that genuinely reaches stage five operates by a different rule: technical elegance applied per business process rather than imposed across them, executed by architects whose depth exists in single digits across the industry. Agentic AI is new enough that the population of qualified practitioners has not yet grown, and most baseline financial systems are not yet clean enough to even pose the stage-five question. Layering agents onto unresolved process, data, and architectural disorder is the

precondition that produces the failures this paper describes. The stage-five exception is real, irrelevant to the strategy decisions facing the overwhelming majority of organizations, and confirms the rule by being so rare it cannot serve as a model for anyone else's program.

---

6. Boundary

This paper has set out the two modes of agentic work in financial systems, the five stages of maturity through which organizations move, and the relationship between maturity and architecture. It has not prescribed a stage-by-stage technology stack, nor named the frameworks, platforms, or tools appropriate at each stage. Those choices depend on the organization's existing data architecture, regulatory environment, and the workloads its agents must serve.

The paper does not address the migration problem of advancing from one stage to the next without halting delivery. That is a separate discipline and a different paper. It also does not address the recognition of regression, the slow loss of maturity through attrition, leadership change, or scope expansion that outpaces the architecture. Regression is real, frequent, and warrants its own treatment.

What the paper hands the reader is a diagnostic. Where is the organization on the curve, in reality. What does the work in front of the team look like, in terms of navigation and execution. Is the architecture currently in place the one that supports that work, or the one that satisfies an aspiration the organization has not yet earned the right to build toward. The diagnostic is straightforward to apply. It requires the kind of honesty about the current state that most programs avoid in favor of articulating the future state they would prefer to occupy. The programs that answer the diagnostic and design for the answer deliver working systems. The programs that produce confident architectural decks and no working software answer for the aspiration and discover the gap when the production system fails to behave the way the architecture promised.