PROFORMA GLOBAL RESEARCH

Does Your AI Agent Need a Semantic Layer?

Matt Rollings, Founder and Principal, Proforma Global

Whitepaper · 2026-06-02

Executive Summary

Background. Agents come in many shapes and sizes, and not all agents require a semantic layer. For example, tightly controlled agents with highly structured process flows that do not require much insight into business language and terminology do not. However, most complex agents and nearly all agents that interface with financial systems require extensive semantic linkages to perform effectively in a scalable fashion. Semantics link organization and business cycle specific terminology to processes and queries, and enable agent-driven actions.

Simple versions of semantic layers are direct descendants of the classical data dictionary. Mature organizations with a fully implemented semantic layer deploy it as an integral part of any data architecture, as the semantics enable the interpretation of multi-dimensional datapoints across various chart segments.

Approach. This whitepaper gives a plain definition of a semantic layer, potential uses, a single test for when an agent needs one, and clarifies the difference between a semantic layer and a simple catalog of governed metrics, which is what most vendors now sell as the semantic layer.

We will articulate what the layer looks like in a finance setting, where the gap between plain English and system meaning is widest, and how to build only the part you need rather than cataloguing the whole business. The engineering and detailed uses for semantic layers are out of scope for this particular whitepaper; the purpose here is to let a leader decide whether focusing on semantic layer buildout sits on their critical path.

1. The Agent That Is Confidently Wrong

A capable model with direct access to your data will answer almost any question you ask it. If an analyst asks for overdue invoices, it returns a list. The issues start when the words in the question carry a meaning specific to your systems, a meaning the model cannot know or infer from its training. Models almost never answer with "I don't know." Ambiguity in definitions will often result in a confidently wrong guess.

Take "revenue last quarter." A general model knows what revenue is and what a quarter is, so it writes a query and returns a number. The model does not inherently understand that your quarter follows a 4–4–5 fiscal calendar that closed three weeks before the calendar quarter ended. It does not know that the figure the CFO reports excludes intercompany eliminations. It does not know that the reporting solution holds both Plan and Actual scenarios of the same line, with multiple versions aligned to the Board, flash, final, and What-if1, which models a potential restructuring. The model simply makes a determination based on the information it has available. The answer is displayed confidently, formatted cleanly, and wrong.

The core issue is not that the model needs to understand the business process, but that it must provide multi-dimensional clarity into the business's systems and data.

Two opposite ends of the spectrum exist without proper semantic layers:

1. A plausible but incorrect answer is returned, creating distrust and causing miscommunication and misalignment.
2. An organization defaults to explicitly asking for every combination of every intersection it wants to report, at the appropriate level. Users abandon the agent rather than spell out every intersection by hand.

Modern AI systems must balance the correctness of output with the intrinsic knowledge that someone in Finance has without needing to explain it to the model on each request. Judging success by whether the output looks right is one of the largest causes of stalled agent programs. Oftentimes the judgment of whether a number is "correct" changes based on context.

---

2. What a Semantic Layer Is, and Why It Varies by Function

Strip away the product categories and a semantic layer is one thing: whatever bridges the gap between what the model understands on its own and what your business requires it to understand. For a model working in natural language, the bridge is mostly linguistic. It is a translation of your terms, rules, and relationships into something the model can read and apply. The form varies with the model. The intent does not.

A plain way to hold the idea: the model arrives knowing the English language and the general world. It has no grasp of your chart of accounts, your close calendar, the difference between Plan and Actuals, or which of three tables labeled "customer" the revenue team trusts. The

semantic layer is where that knowledge lives, in a form the agent can use at the moment it reasons, rather than buried in the heads of the people who built the systems.

The challenge organizations face is that a semantic layer has different meanings and uses depending on the business function. A semantic layer for an HR organization and its processes needs to track different elements than one for a financial system. Financial systems carry concepts such as valid and invalid combinations, business cycles, Management vs GAAP, and allocated vs unallocated, along with countless other alternate views that carry weight for reporting. Each consumer and source system has to be translated into a consumable data model that an agent can parse, understand, and explain.

---

3. The Test for Whether You Need One

Not every concept an agent meets needs translating. The model already reads most of them the way you do, and translating those again would be wasted effort. The remedy for an agent that answers confidently and wrongly is rarely a larger model. It is a careful layer over the few concepts where the model's plain reading and your real meaning come apart. Finding them is the first task, and one test draws the line.

> **The native test.** If the model would reach a substantively correct answer from the question alone, no semantic layer is needed for that concept. If its untutored reading would be wrong or incomplete, a layer is required.

Only someone who already knows the right answer can apply that test, which makes it a judgment for the people who own the meaning rather than for the agent. Deciding whether the model's plain reading is correct takes the very expertise the layer is built to capture. The people who would author the layer are the same ones who decide where it is needed. Sadly, few use cases that reference a multidimensional data model such as a financial system will pass this test.

Two questions show the line. Ask for the largest invoices this month against a tidy table, and the model's plain reading is close enough to the real one. Build nothing. Ask for invoices on credit hold where the dunning workflow has reached stage three and the hold has not cleared, and

the word invoice stops being a generic document. It now means a row in an ERP, carrying system-specific fields and a state machine the model has no way to infer. Build the layer for the second question, not the first.

Scoping follows from the test. Begin with the questions the agent will actually answer. A finance agent might validate an invoice, forecast cash, or explain a variance. Run each one through the native test and keep only the concepts whose plain reading would mislead. Whatever the model handles correctly on its own is work you never have to do. A gap counts only when it changes the answer, so a concept the model reads differently but resolves to the same number is not worth the overhead. The gaps sit at the level of the question, not the system as a whole. Within a single finance agent, vendor and approval may be safe to leave alone while consolidation entity and recognized revenue are not. Build where the gaps are, and only as deep as each question requires.

---

4. The Depth That Is Specific to Finance

Finance is where plain English and system meaning drift furthest apart, and an agent that misreads the gap produces its most convincing wrong answers. A financial figure is rarely a single value. The language of finance is the intersection of business processes and each segment of the chart of accounts. A financial account means little without the additional business dimensions behind it. Semantic layers consolidate business dimensions into consumable relationship groupings for actions and analyses. Semantic layers that do not go far enough to catalog and explain the multidimensional constructs to LLMs are liabilities and data-debt that will eventually be paid in productivity losses.

Valid and invalid combinations. A general ledger will not accept every intersection that is submitted. Revenue may be valid against an operating cost center and meaningless against a balance-sheet entity. A product line may post in some geographies and nowhere else. A controller knows which intersections are appropriate and would never ask for the ones that are not. The model does not know, so it builds a coordinate that parses as clean English, runs it, and returns rows for a combination your books never use. Recording which intersections are real and which are forbidden is some of the most valuable work the layer does.

Scenarios and versions. A financial system stores the same line many times over. Plan sits beside Actual, and Plan alone exists as a Board version, a flash, a final, and however many what-if cases someone has saved. An agent that returns Plan when the user meant Actual, or the flash

when they meant the final, gives back a number correct in every respect but the one that matters, and nothing on its face reveals the swap. Version and scenario need their own treatment, because mistaking one copy of a line for another is easy and silent.

Alternate views of the same number. Management reporting and GAAP describe the same company and seldom agree on a number. Allocated and unallocated cost answer different questions about the same spending. A 4–4–5 period closes weeks away from the calendar quarter, and the revenue a CFO reports strips out intercompany eliminations the raw ledger still carries. Every one of these is a legitimate view, and none is the obvious default. The layer is where a company records which view a given question assumes, so the agent never quietly picks one itself.

None of this is exotic knowledge. It is how FP&A and Accounting folks speak. They process and apply it without being asked. The layer simply writes it down where the agent can read it as it reasons. The list runs long in finance. We make the broader case for treating finance as the hard case in Enterprise Agents in Financial Systems.

---

5. Why a Catalog Is Not a Semantic Layer

Much of what vendors now sell as a semantic layer is really a governed metrics catalog, a certified set of definitions and the relationships among them. Revenue means this, margin means that, and the joins between tables are written down once so every tool computes them the same way. A catalog like that is worth having, and the good ones already carry relationships rather than isolated terms. The catalog is the floor. It is not the ceiling.

A finance agent needs more than the catalog gives it. A catalog can define revenue, but it will not stop the agent from asking for it against an entity where it cannot exist. Those valid and invalid combinations have to be enforced, not merely defined, or the agent builds a coordinate that returns rows and means nothing. A catalog can also define a measure without deciding whether a question wants Actual or Plan, the flash or the final. Choosing the right version is a judgment made the moment the question is asked, and a static catalog is in no position to make it.

Most of all, a catalog defines terms one at a time, and real questions never arrive one term at a time. Asking whether an invoice is valid is not a lookup of the word invoice. The agent has to hold the invoice alongside the purchase order it cites, the contract behind that PO, the balance still open on it, the period it falls in, and the standing of the vendor. Drop any one of those threads

and the answer is no longer safe. A layer worth the name is built around the questions the agent answers, so it can hand over the whole constellation in one piece instead of a stack of separate definitions for the model to reassemble. Assembling the right constellation for each question at the moment it is asked is a discipline of its own, and we treat it in Dynamic Context Assembly.

The history explains the gap. The earliest semantic layers were descendants of the data dictionary, the document that recorded what each field held and which values it would accept. A governed metrics catalog is that same idea brought forward, with certified definitions and the relationships between them added on top. A mature semantic layer goes a step beyond both. It enforces the legal intersections, settles version and scenario, and assembles the constellation a question needs while it is being answered. Neither a dictionary nor a catalog was ever built to do that.

---

6. The Layer Grounds the Process Without Running It

None of that is work a model should do by writing its own query and trusting what comes back. The code that enforces intersections, settles versions, and assembles constellations is deterministic, written and tested in advance, not the model deciding on the spot what to fetch. The distinction is the entire point in financial systems.

The reason is mechanical, not a matter of taste. A deterministic path can be read, tested, and signed off one time, and it runs identically on every request, so the route from question to answer can be audited after the fact. A model that writes its own query brings back the exact problem section 1 described. The result looks plausible, nothing guarantees the path behind it was sound, and there is no way to separate a right answer from a confident wrong one. The layer settles that doubt once and for all questions when it grounds deterministic code. When it grounds the model's own query-writing instead, the doubt simply moves to the one place no one can inspect.

A language model can be trusted with parts of a process that touches money, never with the whole of it. Once the layer has handed it the meaning, the model is good at the narrow, grounded judgments it was given, and it stays poor at running the workflow or checking its own work. The layer is what the deterministic code reads to assemble the right grounding before the model ever sees the question, and what the model reads for the few judgments left to it. The layer is not permission to let the model write its own query and trust the answer. Selling it as self-serve querying that just works puts the model back in charge of the process, which is where the plausible wrong answer came from in the first place.

Two things separate a finance-grade layer from a published glossary. The meaning has to be enforced, not just written down. A definition the agent is free to read and then ignore is not a control, and in a regulated close the enforcement and the audit trail are the entire value, which is the subject of Governance as Middleware. The meaning also has to be authored by the people who own it. What counts as a closed period or as recognized revenue is the finance function's to define, not the database team's, a split we set out in Is AI Part of IT?.

Built this way, the layer keeps the agent clear of the two bad outcomes it otherwise swings between. Users stop spelling out every intersection by hand, because the meaning is recorded once and applied to each request. The agent stops offering a confident answer no one can check, because the orchestration decides what to fetch and enforces the meaning on the way through.

---

7. So, Does Your Agent Need One?

The honest answer is that you need a layer wherever the agent would otherwise be wrong in a way no one catches. In finance that covers most of what matters, though not quite all of it. An agent that reads a single posted balance from one ledger, or lists the open purchase orders above some threshold, stays close to the plain reading and may need little or no layer at all. An agent asked to reason about the close, the consolidation, or the forecast needs a real one, because every such question is thick with the intersections, versions, and views the systems encode and the model cannot guess. The dividing line is not drawn company by company. It runs question by question, which is also the order in which you build, starting with the questions whose unaided answer would be wrong.

The decision in front of a leader comes down to two questions. Does the agent you are funding face questions whose answers depend on what your business specifically means? If it does, has anyone written that meaning down in a form the agent can use and the orchestration can enforce? When the answer is no, the agent will be confident and, now and then, expensively wrong, and no larger model will repair that. When the answer is yes, the semantic layer is on the critical path, and the earlier the gaps are mapped the less they cost to close.

How the layer is engineered from there is a deeper subject, covering the kinds of meaning worth modeling, the way relationships are represented, and how the right context is assembled for each question while it is being answered. We take those up in Semantic Layers in Enterprise Agent Systems and across the Data Architecture for Enterprise Agents series. The trade-off you manage as you build, where finding the right grounding gets easier even as each new process

makes execution harder, is mapped in The Agentic AI Maturity Curve in Enterprise Finance. Let the real questions decide what to build, and the layer grows into the data architecture the business needed all along.

DOWNLOAD PDF

Work with Proforma Global

The thinking in this paper is public; the methods that turn it into a working system are not. If it fits a problem your team is working on, that is what we bring to an engagement. Start a conversation: **info@proforma.global**.