PROFORMA GLOBAL RESEARCH

Self-Learning Configuration in Enterprise Agents

Matt Rollings, Founder and Principal, Proforma Global

Whitepaper · 2026-05-24

Executive Summary

A self-learning agent keeps what it has learned as configuration, not code. It is a store of plain entries that deterministic code reads before the agent acts and writes to only between runs. "Self-Learning Architectures for Enterprise Agents" settled what belongs there. This settles how it should ideally be designed.

A learned rule is only true conditionally. It attaches at a functional level in a business process, across and between dimensions and hierarchies that the business already runs on. That level might be a parent entity, an account grouping, or a product line, and from there the rule governs everything beneath it, much the way a group policy governs the subsidiaries that sit under it. Put a capitalization rule on the consolidation parent and it covers every entity below. A single subsidiary can override it with a rule of its own, and an empty entry can switch it off for a node it should never have reached. Period and year decide when the rule is live, so it fires at quarter-end but not at month-end.

None of this is new. It is how a business already writes policy: a standard is set high and then carved out wherever local conditions force the issue. The hierarchies the agent's rules ride on are the same ones "Semantic Layers in Enterprise Agent Systems" describes. When two rules land on the same case, the business decides which one wins and the architecture enforces that decision the same way every time. The agent does not settle it. New rules go in on the rhythm of the close, at whatever level their evidence and their dollar exposure justify, and each one gets checked again whenever the hierarchy shifts beneath it.

---

1. What the Store Holds

The store is just a set of discrete entries a person can read. Deterministic code reads it before the agent acts and writes to it only between runs, and that code never changes. So the learning is data the agent adds, not edits it makes to the program. The companion paper establishes this separation.

The store holds three kinds of entries:

Rules. A condition paired with an action, run identically on every case it matches. Rules are the most containable item an agent can learn, and the most stable way to concentrate its learning. What a rule will do is visible before it runs.

Patterns. A recurring case the agent has learned to recognize. A pattern decides nothing on its own. It just spots that the case in front of it is one the agent has handled before, and hands it to a rule or a person. That buys speed, at the risk of recognizing too eagerly, a failure the companion treats. A pattern could be a multidimensional combination of Entity-Department-Account that always triggers an autoreversing journal entry at quarter-end.

Antipatterns. The opposite: a case the agent has learned to refuse. When one shows up, the agent holds it, escalates it, or turns it down outright instead of acting on it. Antipatterns are the equivalent of invalid combinations for a specific activity or business process.

---

2. Rules Inherit From the Level They Sit At

A business sets policy from the top down. A group fixes a capitalization threshold or an intercompany matching standard, and it binds every entity beneath until some local rule says otherwise. The business already organizes its numbers this way, through the entity tree, the chart of accounts, and the product and department hierarchies, and the policies that govern those numbers ride on the same structure. A learned rule belongs there too. A rule is really just process logic written down, and process runs from the top of the organization down to the leaf.

So a rule attaches at a branch of a hierarchy, not necessarily a leaf. Set it on a parent entity and it covers every legal entity beneath, by inheritance. The account and product hierarchies work the same way: a rule on a class or category reaches everything that rolls up into it. Period behaves differently. A close phase does not pass a rule down a hierarchy. It only decides when the rule is live, which is how one entry can fire at quarter-end and stay silent at month-end.

Take a group whose quarterly close does work the monthly close does not: intercompany invoices matched and eliminated, capital costs in a construction-in-progress account reclassed into the fixed-asset register. A single rule captures the treatment, attached across three dimensions at once:

- the consolidation parent, reaching every subsidiary beneath it
- intercompany transactions, and the partners valid for that transaction type
- the quarter-end phase, where it fires and nowhere else

A reviewer reads that one entry to see what governs the group's quarter-end treatment, rather than reconstructing the same intent from forty restated copies.

Real hierarchies are not always clean trees. A node can roll up through more than one parent at once, and sorting out those paths is the structural problem "Semantic Layers in Enterprise Agent Systems" takes on. The store is only as good as the hierarchies under it: get the membership wrong, and every rule that inherits through the bad branch inherits the mistake. Different agents apply different rulesets. An agent working on an ERP uses GAAP semantics, whereas an agent operating on an EPM or reporting tool operates on a management reporting semantic layer.

---

3. Override and Exemption

Inheritance alone is too broad. Exceptions are generally the rule in finance. A group capitalization policy holds for most subsidiaries, then breaks for the one sitting in a different tax jurisdiction. The store has to carry that exception at the exact level it lives, or the inheritance that made it efficient turns into a liability. A good example is a private equity group whose portfolio companies operate across different industries and apply different accounting policies to similar expenses.

An override is just a more specific rule that wins over whatever a node would otherwise inherit. The subsidiary that capitalizes on its own threshold carries its own asset rule, and that rule governs it while every sibling keeps inheriting the group's. The level at which someone attaches a rule is what sets its precedence, so the more specific the level, the higher it ranks.

Sometimes the right setting at a node is no rule at all, and that has to be written down, not left blank. An empty entry turns the inherited rule off there on purpose. A blank would just let it flow down. An auditor cannot tell an accident from a decision unless someone wrote the decision down. That is why the exemption has to be an entry of its own.

A rule that attaches across several hierarchies can be reached by more than one at once, say one coming down the entity tree and another down the product line. Which one wins is a business judgment, not a technical default. Statutory entity rules might outrank product-line rules in a regulated industry. Somewhere else, the priority flips. Either way the business sets that precedence and the architecture enforces it the same way every time. The agent never settles it at runtime, which is exactly the line "Governance as Middleware" draws between deciding policy and carrying it out.

All of this keeps the damage contained. The agent can add a tight rule at one node without touching the group rule above it, and if that exception turns out wrong, it is one entry to pull while everything else stands. A learned rule reaches as far as the level it hangs on and no further than that, which is what makes a bad one cheap to undo.

4. The Line Between a Rule and a Fix

A proposed rule has to earn the name before anyone writes it down. A real rule comes with the concrete cases that prompted it and a plain statement of why it exists. Take those away and what is left is an abstraction, and an abstraction tied to no real cases will not last a quarter in production. The cases matter later, too. They are what a reviewer reads a year on to decide whether the rule still holds, and without them a rule cannot really be governed at all.

Often those cases reveal that the proposal was never a valid rule. An agent reaching for the same fix again and again is usually flagging an inconsistency in the data upstream or downstream, or a process so fragmented that no clean rule fits it. The recurring need to paper over a case is an important management decision, not something to encode. Stored as a rule, the patch spreads across the hierarchy while the break it hides stays in place.

So the first question about any candidate is whether it is really a rule at all, or just a fix for a data, process, or technical problem dressed up to look like one. Only a genuine rule belongs in the store. Drawing that line before anything goes in is what keeps the store compounding in value instead of filling up with workarounds. What those misclassified workarounds cost, and why an agent cannot be trusted to draw the line on its own, is the subject of "Learning Instead of Fixing."

5. Attaching at the Right Level

Once a proposed rule is approved through a governance mechanism, it is attached at a certain level of the process and hierarchies. The proper level of scrutiny must be applied to each rule, with the thoughtfulness scaled to the materiality of the dollars the rule governs.

A rule at the group parent governs the summed exposure of every subsidiary beneath it, so a ten-thousand-dollar invoice rule attached there can move millions in aggregate. That aggregate, not the size of any single case, is the real risk "Self-Learning Architectures for

Enterprise Agents" names.

The configuration exposes that risk by level, meaning the governance it enables has to be deliberate. Organizations must develop a rigid process to evaluate each rule, define the documentation required to approve a rule, and ensure it is monitored. Rules can cross any type of activity at any level of detail, and due to the formlessness that rules can take, teams must add additional process checks at each level of agentic operations.

---

6. The Cadence of the Close

The close calendar sets the clock for all of this. A learned rule earns trust only by surviving the cycle its condition belongs to. Each proposed rule should be evaluated and decided on the slowest cycle its validation requires, which the close lays out in three horizons.

Monthly. The smallest complete cycle, and where the narrow rules live. These are the within-period cases scoped to a single entity, account, and phase, the ones that resolve the same way every close. Once a candidate has held across several monthly closes at one node, it can be absorbed there.

Quarterly. Consolidation, intercompany, and the forecast only resolve across a full quarter, so any rule that depends on them cannot be validated sooner. The quarter is also when a rule that has proven itself at several sibling entities gets weighed for promotion to their shared parent, since only a consolidation shows whether it really holds across the group.

Annual. Year-end settles the structure itself. The work then is mostly testing the rules already in the store against hierarchies the year may have moved underneath them.

That cadence is also a review, not just an intake. Each quarter or year, someone reads the whole standing set and decides, rule by rule, what to keep, refine, or retire, and which new candidates have earned a place for the period ahead. You can only review a set like that because each rule is a single entry a person can read, scoped to a level. This is where the configuration meets governance: the review hands down a verdict on each rule. Who gets to issue that verdict, and under what controls, is the governance question this paper leaves to one side.

One principle holds no matter how a given close is built. A rule goes in no faster than the cycle that can validate it. It moves up the hierarchy only after the cycle that proves it belongs has actually closed.

7. When the Hierarchies Move

The hierarchies a rule inherits through evolve and change. Anyone who has run a close can name a dozen rules that were valid one year or period and wrong the next. As organizations begin to understand how business processes, data, and activities change, learning configuration and approval mechanisms become increasingly critical. Self-learning architectures become overhead rather than adding value without them.

The risk is almost always process-related, not technical. Technology cannot adapt to changing business conditions or processes without a definitive control framework.

No matter how powerful an AI is, the volume of information it would need to process to make fully autonomous decisions on its own is nearly impossible to manage. However, the better question is whether AI can accelerate the understanding of process gaps that are otherwise hidden, creating a tighter fit between systems and process.

Inheritance is where that fit is won or lost, and it cuts both ways. One change at a parent rewires every rule that flowed through it, but clarifying the rules as global policies that require enforcement is a powerful tool.

The real challenges come from structural changes to the organization and its hierarchies, and the downstream impacts on its governance processes. When a leadership change drives a geographic realignment of cost centers and profit centers onto a new responsibility matrix, how do bonus accruals change? Which rules are now invalid? Similarly, from an accounting perspective, how does splitting a legal entity change the rules and the history? Asymmetrical data across relevant time horizons can paralyze agents or send them into a catastrophic analysis loop. Technically, the rules still read as correct, but they now govern a population their originator never chose. Failures in these cases are silent and late.

Catching issues before agentic processing reaches the financials is critical work, and someone must own it. Handling structural policy changes will create a new type of integration specialist, one who ensures coherence within the business and sees that policies are bridged appropriately, documented, and understood.

What that specialist does has a name. Keeping a layer aligned as the business changes is the discipline "Semantic Layers in Enterprise Agent Systems" sets out, and a self-learning store carries it further, because its rules are defined entirely by where they hang in hierarchies it does not own.

8. Boundary

This is a position on configuration. Several things sit deliberately outside it.

Governance and change management are out of scope. Who may admit or promote a rule, under what authority, with what audit trail, is the control layer wrapped around the store, and the companion handles that. How a rule gets in is a separate concern from how the store itself is built.

The resolution order is also left unspecified. When several inherited rules reach one case, the precedence the business configured still has to be computed somewhere, and that is engineering this position implies without working out.

Then there is the learning the agent folds into its own reasoning, the instructions and examples it weighs rather than obeys. The companion takes that up. This paper is the deterministic half of the pair.

Drift detection sits outside the boundary too. Catching that a rule has gone stale, or that the hierarchies have shifted underneath it, means measuring the agent's output against a standard it had no hand in setting, and that is a discipline of its own.

The last thing held back is the cost of the non-rules, and what happens to a system when an agent quietly absorbs them on its own. Those questions, along with the case that most of what an agent learns is really deferred process, data, and technical work, belong to "Learning Instead of Fixing." Here the store has just one job: keep out what is not genuinely a rule.

A self-learning agent's knowledge is a store of rules, patterns, and antipatterns, read and written by deterministic code. Each entry hangs at a level in the hierarchies the business runs on and inherits down from there, until a more specific level overrides it or switches it off, and the whole set is absorbed and re-checked on the rhythm of the close. The store inherits along the structure the business already uses, so it can still be read end to end however large the business gets. Every rule is a single entry sitting at the level someone chose for it. The flat alternative, one rule restated per entity, drifts apart until no one can say what governs the group.

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**.