PROFORMA GLOBAL RESEARCH

# Dynamic Context Assembly

Matt Rollings, Founder and Principal, Proforma Global

Whitepaper · 2026-05-18

## Executive Summary

Dynamic context assembly (DCA) builds the model's input window one request at a time, drawing from a structured data model and the supporting structures that make selective composition possible. DCA keeps the model's attention from degrading. As a context window fills, reasoning quality drops regardless of the model's stated token limit. The assembly step composes the smallest set of inputs sufficient for each reasoning step, deterministically, from the components that data model defines. This paper sets out the practice, the problem it addresses, what a workable data model has to solve, and where the discipline stops. The design concepts and best practices behind each component come in the papers that follow.

## 1. The Problem

Language models degrade as their context windows fill. A million-token window accepts a million tokens and reasons worse the closer the prompt gets to the limit. Relevant content gets lost in the middle while irrelevant content competes for attention and pulls outputs toward whichever fragment reads as salient.

Retrieval-augmented generation does not solve this. Chunking a corpus and pulling top-k similarity matches still fills the window with content of unknown authority and uncertain applicability. The model does the filtering work that should have happened before the call.

Enterprise agents make it worse. They bring internal and ambiguous vocabulary, data with structure and freshness and permission constraints the model cannot infer, and rules the output has to conform to. Every addition is at once material the model needs and volume that degrades its attention.

Dynamic context assembly composes the smallest set of inputs sufficient for the specific reasoning step in front of the model. The data model is what makes "sufficient" defensible.

## 2. What It Is

Dynamic context assembly constructs the data model the language model consumes when it answers a request. The data model is built fresh for each request, composed from the structured foundation the organization maintains for the purpose, and holds the smallest set of inputs sufficient for the reasoning step the model is about to perform.

Earlier practice went at the same problem one prompt at a time. A human author drafted instructions, examples, and constraints into a static prompt, refined the prompt against observed outputs, and accepted the result as a fixed artifact. The practice was called prompt engineering, and it remains the dominant approach in production agent deployments. It produces brittle artifacts that resist measurement, accumulate without coherence, and degrade as the agent's scope expands beyond what the original author had in mind.

Retrieval-augmented generation tried to make the prompt dynamic by appending retrieved fragments at request time. The mechanism works for narrow factual lookup against a single corpus. It does not survive enterprise scale, because retrieved fragments arrive without authority, freshness, permission, or topology, and the model is left to filter and reason at the same time over inputs of unknown trust.

Dynamic context assembly replaces both. The organization builds a structured data model whose components are designed to be selectively composed. An assembly step runs between every request and every model invocation. It composes the appropriate slice of the data model for that request, records what it composed, and calls the model against a context window built for the question at hand. The composition is per request. No two requests see the same window.

Most of the difficulty of DCA is not in the assembly step. The assembly step is mechanical once the data model is right. The discipline is hard because the data model has to be designed correctly, and the data model is the subject of the rest of this paper.

## 3. What the Strategy Has to Solve

Organizations that try enterprise agent deployment without first sorting out their data architecture get the same outcomes regardless of vendor, model, or framework. The agents are convincing in demos and unreliable on real workloads. They look promising alone and come apart in combination. The business stops trusting them before the program reaches scale. The pattern is consistent enough to be diagnostic. Neither the model nor the orchestration layer causes this. The agent simply has no data model to reason against.

A workable data model has to solve several problems the organization usually has not solved at the level the model requires.

**Resolution of ambiguous vocabulary.** Business terms carry meanings that depend on context the user never states. In finance, Net Income means substantively different things depending on whether the data is pre- or post-elimination, actual or plan or forecast, before or after topsides, this version or that one. A human reader resolves the ambiguity from context. A model cannot, unless the data model has resolved the term before the model is invoked.

**Reduction of data volume to the slice the request needs.** Models are not helped by being shown the full corpus. They are degraded by it. The data model has to know which slice serves which kind of question and produce that slice on demand. The selection has to be deterministic, because non-deterministic context produces non-reproducible reasoning, and an enterprise will not trust an agent whose answers shift when its inputs shift out of sight.

**Attribution composed alongside the data, not bolted on.** Every value the model sees should arrive with its source, its freshness, its lineage, and the permission boundary that governs who may see it. None of these properties can be inferred from the values themselves. They have to be present at the moment of the call, from a data model that has carried them since the day the data was first ingested. Organizations that try to retrofit attribution after the data model has accumulated content without it find the work is roughly the scope of the original build.

**The relationships between concepts have to be traversable.** A reference to one entity usually implies relationships to others, including calculation dependencies, dimension members, hierarchical parents, and derived versions, that the model needs in order to reason but cannot discover from the entity alone. The data model has to make that topology explicit and queryable, so the assembly step can compose the whole relevant neighborhood and not just

the named entity. The most common silent error we see in production traces straight to this gap. The agent returns a number that looks correct and is wrong in a way the user cannot catch from the answer.

**Precedence rules the organization decides once and applies consistently.** Which source wins when two systems disagree about the same fact. Which definition applies when the user's role implies one context and the workflow implies another. Which version of the data model the agent reasons against today versus a month from now. All three are organizational decisions, not technical ones, and the data model enforces whatever the organization has decided. An organization that has not decided cannot build a data model. It can only build the appearance of one.

We have not seen an enterprise solve all of these at once. We have seen enterprises solve enough of them to make agents reliable for one class of work, then extend the data model from that foothold. The starting point matters less than the discipline of treating the data model as the load-bearing artifact and the model as the consumer of what it produces.

---

## 4. Semantic Layers

Humans understand information in context. A model takes in everything at once and weighs it all together, which is not how a person reads and never will be. For an agent to interpret data and act on it, the semantic layers it reads should be both multi-dimensional and hierarchical, with context established through inheritance across layers and within them.

Semantic layers exist to turn ambiguous business vocabulary into structured queries before the model is invoked, so the model never has to guess what the user meant when it answers.

In the implementations we have seen, organizations still mostly lack a working understanding of how to organize semantic layers in a way a model can use. Our view is that the hierarchy and the multidimensional relationships those layers carry are the data model an agent needs in order to reason.

Each organization and each implementer will have a view on:

1. What the data model should look like
2. The relationships between semantic layers
3. Inheritance within and across semantic layers
4. The technological mechanism by which the data model is stored and transmitted

No one view is simply right or wrong. Which is correct depends on the workload the agent is being built to serve. What stays consistent across engagements is that without a clear, statable design strategy for the semantic complexity, the ROI on non-trivial agent work stays strongly negative.

An agent that reasons over resolved values without access to the underlying topology cannot answer questions that depend on structure rather than value, and many of the questions a competent business user asks fall into that category. The agent shows no sign of it. A figure that reads as correct turns out wrong in ways the user has no immediate way to detect.

A new kind of master data management is emerging from this work. Traditional master data discipline manages the entities, the customer, the account, the cost center, the product. The discipline DCA requires manages the *relationships* among them, the calculation dependencies, dimension broadcasting, proportionality structures, hierarchical inheritance, version lineage, and the other topology that the values themselves never show. Organizations that build agent programs without seeing this distinction discover, usually about eighteen months in, that they have been treating a relationship problem as an entity problem.

---

## 5. Boundary

Dynamic context assembly decides what the model sees. It does not decide what the model does with what it sees, what verifies the output, what executes the action the agent recommends, or what routes the work according to risk tier. Orchestration, verification, deterministic execution, and risk-tier routing remain necessary parts of the broader agent architecture. A well-constructed data model in front of a broken execution layer still produces well-informed wrong actions at scale.

The discipline also depends on substantive data work the organization has to do before assembly produces value. A semantic layer can only resolve terms the organization has actually defined. Inheritance operates over relationships that have been mapped explicitly between business concepts and the underlying data. Attribution requires source records the organization has been disciplined about maintaining. An organization that tries dynamic context assembly without first doing the underlying data architecture work will produce an expensive abstraction over the same problem it started with.

The remaining papers in this arc describe the data architecture that makes DCA work in production. They cover how semantic layers are organized hierarchically and dimensionally to support inheritance, how attribution and provenance are encoded so the agent's conclusions stay bounded by what the data supports, how the data model evolves as business processes and definitions change, the measurement discipline that tells a data model that is improving the agent's outputs from one that is only producing more elaborate context, the structural limit past which the language model is no longer needed for a class of queries, and the economics that govern scaling as agent workloads expand. Each is a separable concern, and each is required for the foundation defined here to bear production weight.

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**.