

GOVERN MEANING

Ontology is the
invisible infrastructure
that defines reality
for AI, agents, and people.

Who decides what something
means, decides how it will be
reasoned about, automated,
and governed in the future.



// ai strategy papers

ideas, strategies, and impacts
of artificial intelligence

Ontology and knowledge graphs: why your company needs them (and how it's done)

Everyone talks about ontology, almost nobody governs it. And yet it is the structure that decides whether your AI reasons or guesses, whether an answer is defensible before a regulator or merely plausible. A long, considered guide: what an ontology really is, why you need one now, how you build it without reinventing it, and how you keep it from degrading in silence until it turns dangerous.

Take the word "customer." In a meeting where sales, finance and support are all sitting at the same table, that word means three different things at once. To the salesperson it is anyone who has entered the pipeline, even just a contact who left an email address. To finance it is a billed entity, with a VAT number and a payment history. To support it is someone entitled to open a ticket. Three definitions, three departments, one single word, and every report built on that word carries the ambiguity inside without declaring it.

As long as the people reading those reports are human, the misunderstanding gets absorbed. An analyst knows which "customer" a colleague is talking about, adjusts on the fly, asks for confirmation. The problem arises when we put a machine in charge of reading the data, a machine that has to answer, decide, act. The machine does not adjust on the fly. It takes the definition it finds, applies it with confidence, and hands you back a retention figure, a risk exposure, a list of recipients for a campaign, without knowing that the definition it used had two others folded inside it.

This is where the whole story begins, and over the past few months it has ended up on every slide, in every pitch, inside nearly every conversation about how to bring artificial intelligence into a company. The word going around is "ontology," accompanied by its companions, knowledge graph, semantic layer, graph of knowledge. They get uttered as if the meaning were obvious and shared, and it almost never is. You could [sum it up like this](#): now everyone has an ontology, and almost nobody has an ontology.

I want to take that sentence seriously, and explain to the people in a company who have to decide, not just implement, what sits beneath these words, why the subject is exploding right now, and what changes in the way we work when ontology stops being a buzzword and goes back to being what it is, the structure of meaning on which everything else rests.

01 The hollowed-out word

I often play a game, at the tables where I work, when someone brings out the word ontology with the air of having found the solution. I ask: which problem, exactly, are we solving. Nine times out of ten the answer is a second buzzword in place of a definition, we need it for the AI, we need it to put order in the data, we need it because a competitor is doing it.

The same scene, [seen from inside a workshop](#), gets told by the people who live inside software architecture: someone names the AI, someone else talks about APIs, and three concepts that have nothing to do with one another end up mixed together on the table, Canonical Data Model, ontology and knowledge graph, which, when you ask whether they are the same thing, are not even remotely the same, and the confusion has a precise cost, over-engineering, systems that try to do everything at once and end up doing nothing well.

The Canonical Data Model is exactly that, a canonical data model, and it exists to make different systems talk to one another by standardizing the formats of exchange. It lives in the world of integration, not the world of knowledge. An ontology is a different thing, the formalization of the concepts of a domain, of their properties, of the admissible relations and of the rules that constrain them. The knowledge graph is a different thing again, that structure filled with real data, the concrete network of instances. Confusing them has a concrete cost, projects that start off crooked and discover they are crooked months later, when the bill has already come due.

The anti-pattern even has a name, the one model to rule them all, and it describes the temptation to build a single universal model that absorbs integration, semantics and knowledge, and ends up doing none of the three well. Against that temptation I keep a question in my pocket, which is the cheapest form of due diligence I know, and I repeat it in every meeting where someone proposes to solve everything with a knowledge graph: which problem are we really trying to solve. If the answer is a list of three different things at three different levels, you probably need three tools, not one model trying to be all three.

Then marketing shows up, and the fog thickens. Under the label of ontology you can buy just about anything today: Microsoft with Fabric IQ, Palantir with its Ontology, and after them SAP, Salesforce, AWS, each with its own version of the same narrative. Almost always, behind the label, you find a governed property graph, or a business glossary with a few kinds of relation, or a typed schema with some constraints. Useful things, sometimes very sophisticated. But not ontologies in the logical sense of the term, and the distinction weighs enormously the moment an autonomous agent begins to reason over it to make operational decisions.

The difference lies in what is missing, and that is where the risk hides. A formal ontology, in the full sense, commits to a logical semantics made of axioms, takes an explicit position on how to treat what has not been declared, distinguishes individuals from classes, and above all it performs inference: from what you have asserted an engine derives what you never wrote, in a bounded, decidable, verifiable way. [Look inside the vendors' offerings](#) with this lens and the picture changes. Fabric IQ grows out of a Power BI semantic model designed by business experts with no-code visual tools. Palantir's Ontology defines object types, link types and action types, and it is the most sophisticated of the group, but underneath it has no description logic, no inference, it makes no formal commitment to an epistemological regime. The result is a well-governed conceptual map, the promise is semantic reasoning, and the two do not coincide. The stakes are not academic: that same layer gets positioned as the foundation for agents that decide autonomously, it underpins defense operations and critical infrastructure, it connects the data of an electrical grid. An agent acting on its own over an informal semantic layer reasons only as well as a hand-built model is complete, with no inference and no formal validation of the constraints. You cannot certify an agent's behavior against constraints you never specified, and an infrastructure built on a bet of semantic convergence stays fragile and opaque, and fails in silence until it fails all at once.

The question to put to whoever is selling you an ontology, then, is blunt, and I keep it ready for the next meeting: what, exactly, can your system *prove*. Not what it can represent, not what it shows in a demo.

What formal guarantees does it offer when an agent derives a conclusion nobody wrote. If the answer is vague, they are selling you a well-drawn map, not a reasoning engine. It may be perfectly fine anyway, but it is worth knowing which of the two you are buying, because the price and the risks are different.

02 Ontology, taxonomy, knowledge graph

Let's put things in order from the bottom, climbing by degrees, because every rung of this ladder is a concept that pitches skip and that in fact decides everything.

The elementary brick is called a [triple](#), and it comes from the [W3C's RDF standard](#), born to represent information on the web in a machine-readable way. It has three parts, subject, predicate and object, as in "Mario, lives in, Italy" or "Product A, has feature, waterproof," and with these three elements you get an atomic statement, precise, queryable. Put millions of triples together and you have something that resembles knowledge, provided they use the same identifiers for the same things, and here a detail creeps in that looks technical and is in fact political: IRIs, the unique identifiers that keep you from collapsing into the same node the "Paris" that is a city and the "Paris" that is a person. It is the same work as semantic annotation, marking the word "Paris" in a text as a city and not as a person, and it separates the data a machine crosses at random from the data it knows how to read. Around RDF orbit other standards worth naming once so as not to confuse them, SKOS for organizing vocabularies and taxonomies, OWL for real ontologies with their rules, and the linking of different datasets that goes by the name of linked data. These are the bricks with which the semantic web promised, twenty years ago, to make data comprehensible to machines, and they come in handy now for a reason that did not exist back then, giving a language model something solid to stand on instead of the void.

Above triples sits the taxonomy, the hierarchy, the tree of categories, the "is a kind of" relation. A dog is a kind of mammal, a mammal is a kind of animal. The [clearest lexical distinction](#) I have found I borrow and keep. The taxonomy is the hierarchy. The ontology is the hierarchy plus the relations plus the rules plus the constraints on what can connect to what. A fracture concerns a bone, and that is a rule that lives in the ontology, not in the taxonomy. The knowledge graph is the ontology populated with real data, the millions of actual instances flowing through your systems.

The abstraction ladder

From raw data to governing meaning. Each step adds a capability the one below lacks.

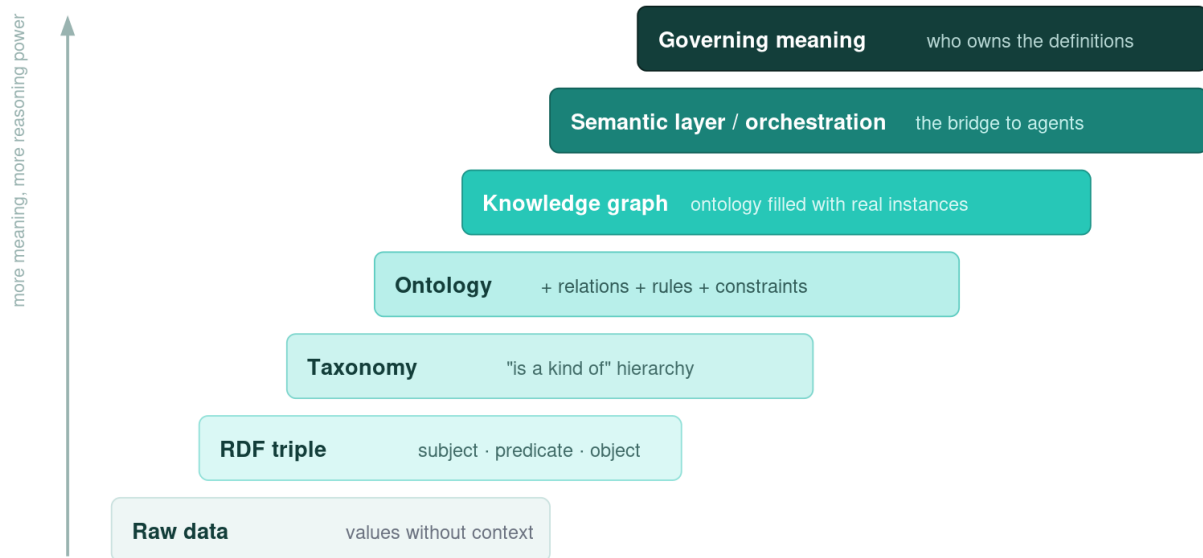


Fig. 1 – The ladder of abstraction, from raw data to the governance of meaning.

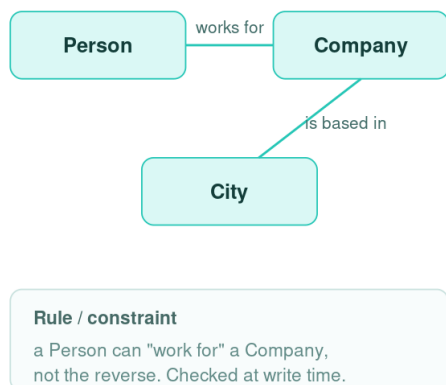
The classic definition of ontology, the one everyone quotes and few have read in full, was given by Tom Gruber in the nineties: [an explicit specification of a conceptualization](#). It sounds abstract and says something simple and powerful. An ontology is the instruction manual with which a machine understands what we are talking about. The manual breaks down into five components worth keeping in mind: the classes (Person, Company, City), the concrete instances (Mario Rossi), the properties (age 35), the relations (works for), the constraints (the founding date must precede the dissolution date). These five pieces are enough for a machine to stop seeing text and begin to see structure.

Ontology and knowledge graph

The ontology is the schema, the graph is that schema filled with real data.

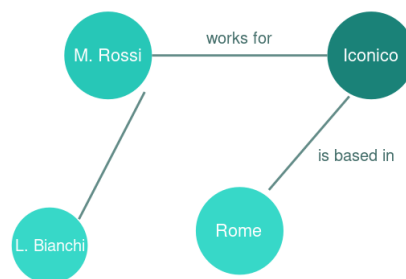
ONTOLOGY · the schema

types, allowed relations, rules



KNOWLEDGE GRAPH · the instances

the same types, filled with real data



millions of typed nodes and edges,
all conforming to the schema on the left.

Fig. 2 – The ontology is the schema, the knowledge graph is its instances.

One is still missing, and it is the one that separates those who have studied the subject from those who merely name it. An ontology carries commitments inside itself, ontological commitments, decisions about what exists, about how the identity of a thing persists over time, about what counts as a primitive relation. The most-cited example in this field I find illuminating for anyone who has to model business data. A person is a type, a kind: it supplies identity, it is rigid, it exists as such. A student is a role: it depends on context, it is defined from outside, and a person can stop being a student without ceasing to exist. Modeling the two as if they were the same category produces, months later, confusion in the data lifecycle, interoperability failures, that silent drift of meaning you discover only when it is too late. A logically valid model can be ontologically wrong, and that is a sentence anyone bringing AI into a company should print out and pin up somewhere.

Above everything sit the upper ontologies, the foundational ontologies that answer a single question, which structural commitments are needed for a representation to count as coherent knowledge of the world. Several exist and they do not agree with one another. BFO, strongly realist, describes reality as it is and serves as the backbone of half of biomedicine. DOLCE looks at how humans conceptualize the world. UFO, with its OntoUML language, holds formal ontology and software engineering together. More than competing vocabularies, they are competing metaphysical commitments. The decision-maker does not need to choose between BFO and DOLCE, they need to know that the choice exists, that someone is making it on their behalf, and that making it badly does not throw an error on screen: it gives you a system that works until it works no more.

One example only, to make the weight of these commitments felt, on a question industry faces every day. Version A of a product, version B, the system under maintenance and the decommissioned one, are they the same thing, or not. One philosophical tradition, endurantism or the three-dimensional view, says the object is wholly present at every instant and change is represented as a succession of states, with time left outside. Another, perdurantism or the four-dimensional view, says the entity extends through time as it does through space, and "yesterday's car" and "today's car" are temporal slices of the same entity

extended in time. It sounds like a seminar debate, and instead it decides how you handle identity, traceability, digital twins, versioning, legal liability, causality. Most industrial initiatives skip this question and use the ontology as operational middleware, and it holds up until the systems evolve and the scales begin to interact. Ontological clarity should precede formalization, not follow it as an afterthought, and that is the exact opposite of how people usually proceed, where first you build the graph, then you compute the embeddings, and you deduce the semantics at the end.

There is also a technical price worth knowing, because it explains why many things here are harder than they seem. OWL, the most widespread language for formal ontologies, in its tractable variant is designed around decidability: it wants an automatic engine always to conclude a reasoning process in finite time, and to guarantee that it gives up a bit of expressiveness. Meta-modeling becomes difficult, contextual semantics weakens, self-reference is constrained. These are the price of decidability, not defects, and it is often worth paying, provided you know you are paying it. Fifty years ago Arthur Koestler coined the word *holon* to indicate something that is at once a whole and a part, and this double nature throws clean decomposition into crisis, along with sharp boundaries and hierarchies that contain one another tidily. Organizational reality is made of holons, of things that are a department and a node of a larger network at the same moment, and flat graphs, made only of nodes and edges, simulate that structure by convention, accumulating a semantic fragility you discover when fixing it costs dearly.

Having stated the limits, there remains a technical reason why ontology and graph marry well, useful to know because it explains why relational databases struggle precisely here. In a graph every node physically holds the pointers to its relations, so traversing an edge is a direct jump, of constant cost, independent of how large the graph is. In a relational database the same question, who is connected to whom two, three, four steps away, becomes a chain of joins that multiply and degrade fast, that misalignment between how we think about relations and how a table stores them that computer scientists call the impedance mismatch. And then there is real reasoning: over a typed graph with an ontology on top, transitivity becomes automatic, if A manages B and B manages C then A manages C even if nobody wrote it, and so does inheritance, a query on "employee" also picks up the managers if manager is a subtype of employee. These are answers to questions never asked, and they distinguish a structure that reasons from an archive that merely returns what you put into it.

03 Structure as the frontier

For twenty years the watchword in data projects has been quality. Clean data, reliable data, verified data. Correct, and not finished, but the constraint that blocks the value of AI today has shifted up a step, and almost nobody is looking at that step. [Data quality](#) tells the organization whether a piece of information is reliable, data structure tells the machine what that information means and how things relate to one another. Two different questions, and reasoning lives in the second.

Inside there is a subtle and decisive distinction, between metadata and structure. Metadata describes what a document is, and it is enough to find it again. Structure makes explicit how things relate, and it is what you need to reason over them. You can have a perfectly catalogued archive, every file with its tag, and still be unable to answer a simple question like what is our overall exposure to a certain country, because that answer sits in no single document, it sits in the relations among suppliers, contracts, jurisdictions, raw materials, and in the well-catalogued archive those relations exist nowhere, except in the head of the person who reconstructs them by hand over two weeks.

Let's follow that question all the way, because it shows the difference in physical terms. Supplier data, in flat tables, is enough to pay an invoice: an identifier, a name, a contract value, some terms, a contact. It is not enough for exposure to a country, because that question lives in the connections. In a graph, every supplier, every jurisdiction, every raw material, every contract, every regulatory framework is a typed node, and the question becomes a path: you start from the country, reach all the suppliers operating there, then the connected raw materials and contracts, then the suppliers of the suppliers, and you read the concentrations that form along the way. A path through the graph, generated when needed, and not a spreadsheet assembled by hand over two weeks. The same structure that makes the question possible makes it defensible: under a flat retrieval you produce the policy that should have applied, a defense built after the fact; under a graph you reconstruct the specific decision that was made, a record. Quality makes an answer correct, structure makes it defensible.

Quality and structure

Quality gives reliable nodes, structure gives the relations. You reason, and defend, only where both are present.

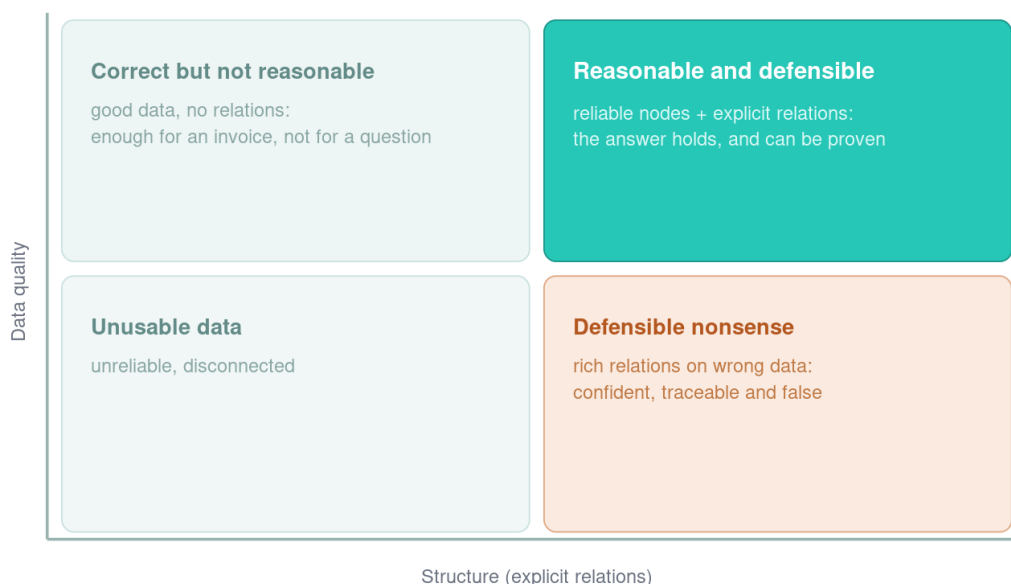


Fig. 3 – Quality and structure: you reason, and defend, only where both exist.

It is not a 2026 fad, and you can tell from a story that spans thirty years. In 1998 there were people digitizing regional newspapers, cataloguing every article with fifteen fields of a metadata standard, the Dublin Core, and from that metadata came the first revenue stream, because content could finally be found again. Then every article was linked to a directory of businesses, the car dealer, the real estate agency, the headhunter, and from those relations came the second revenue stream. It was a knowledge graph without today's vocabulary, and the sequence is the same one we are living through at company scale, first no structure, then the metadata, then the domain ontology, then the graph extended across the whole enterprise. The jump in scale is enormous, the logic is identical.

For decision-makers an image works better than any diagram. The chart of accounts defines what is revenue and what is cost, and no serious company leaves it to chance, it guards it, governs it, defends it. The definitions of the entities in a knowledge graph, what a customer is, what a product is, what a supplier is, carry today the same weight as the chart of accounts, but over a much wider surface, because they touch everything downstream: the numbers you report, the obligations you run into, the people an AI system will treat as the target of a decision. The chart of entities is the new chart of accounts.

From there descends a concept that should make leaders' antennae go up, the verification premium. An organization can verify only what its systems know how to reason over, and at scale reasoning is a function of structure. There are four kinds of indicator: lagging ones, that state a fact already occurred and for which reliable data is enough; anticipatory ones, that read a signal; predictive ones, that estimate a probability; and reasoned ones, that derive a conclusion from premises. Only the last requires the ontology, and only the last cannot be falsified by adding more data. You can have all the data in the world, but if you cannot reason over it in a traceable way, you cannot prove that your conclusion holds.

There is a formula, rough and true, that reaches boards directly: [you cannot scale AI without scaling semantics](#). The ontology, read this way, is a living, executable schema, and not a static document hung on a wall: when you query the system you are not querying the raw tables, you are querying its view of the world. It takes concrete form in three elements that become rules applied at write time, not good intentions. A type system, the labels on the nodes and the types on the edges, taken from the class definitions. A property schema, what is mandatory and what optional for each type, checked when the data comes in. And constraints on the relations, which links are admissible between which types, a customer can place an order, a supplier cannot, with the rule applied on insertion or on query. The marketing "customer," whoever left an email, and the finance "customer," whoever completed a paid transaction, stop diverging, because the ontology creates a single internal source of semantic truth, the official account of how that enterprise defines its own things, and the graph enforces it at the level of storage and of querying. It is not a universal truth, and I will come back to this later, it is the coherence of an organization with itself, which is already far more than most companies possess today. Semantic stability becomes a property of the system, not a wish.

[Data without semantics is noise](#), said from the side of those who build graph databases. Having petabytes scattered across dozens of systems, each with its implicit identifiers and its undeclared assumptions, does not make you data-driven, it makes you a company that has a lot of data and little knowledge. Structure turns the noise into something a machine, and before it an organization, can reason over. And that structure, in implicit form, already exists inside the company. The latent graph is there, in people's heads and in the joints between systems. The choice is to make it explicit and govern it, or to let it form on its own.

04 Grounding against hallucination

We come to the real reason, the one that has brought ontology back to the center after years as a topic for academic conferences. The reason is called the language model, and the most honest way to put it is that an LLM is extraordinary at producing plausible text and is not designed to be correct. Two different qualities, which we confuse constantly because plausible text, while we read it, looks correct to us.

The models are extraordinary [pattern recognizers](#), with no intrinsic understanding of truth, and that is perfectly fine for many things and not at all for others. A "probably correct" is acceptable when you suggest a film, not in manufacturing, in the supply chain, in fraud prevention, in medicine, where a wrong answer stated with confidence costs money, compliance, sometimes lives. Here two strands of artificial intelligence kept apart for decades come back together, the symbolic one made of rules and structure and the statistical one made of learning from data. The ontology brings the structure, the model brings the flexibility, and together they give what neither gives alone, a flexible system that nevertheless knows when it is saying something true and when it is guessing. Grounding, the anchoring of the model to an external, deterministic body of knowledge, is the bridge between the two worlds.

The sequence to learn by heart is that [the model should be used as the last step, not the first](#). If you use it to reason over a structured domain, products, patients, bank accounts, legal policies, you are asking the right thing of the wrong tool. The reasoning should happen first, over the structure, and to the model you leave the last mile, putting into language an answer that has already been verified. Without ontology you have data, with ontology you have meaning: the ontology gives the meaning, the knowledge graph gives the true facts of right now, and neither one, alone, is enough.

Anti-hallucination plays out on a more concrete plane than people believe, and you understand it with a trivial example. The feature "waterproof" is valid for a jacket, meaningless for a pair of headphones. A model left to its own devices, if you feed it fragments of text about headphones and about waterproofing, stitches together a fluent sentence in which the headphones become waterproof, and hands it to you with the confidence with which it would tell you the sky is blue. A three-layer architecture stops exactly this. Layer one is the triple, the bare fact. Layer two is the constraint, written in OWL: the property "has feature" has a domain and a range, and a triple that violates them gets blocked there, before it even reaches the model, and that is where the semantic guard lives. Layer three is the linguistic frame, the deterministic template that turns the verified fact into a readable sentence. The sentence that comes out is resolved, not generated.

The three layers that stop hallucination

The sentence that comes out is resolved, not generated. The semantic guard blocks the invalid fact before the model sees it.

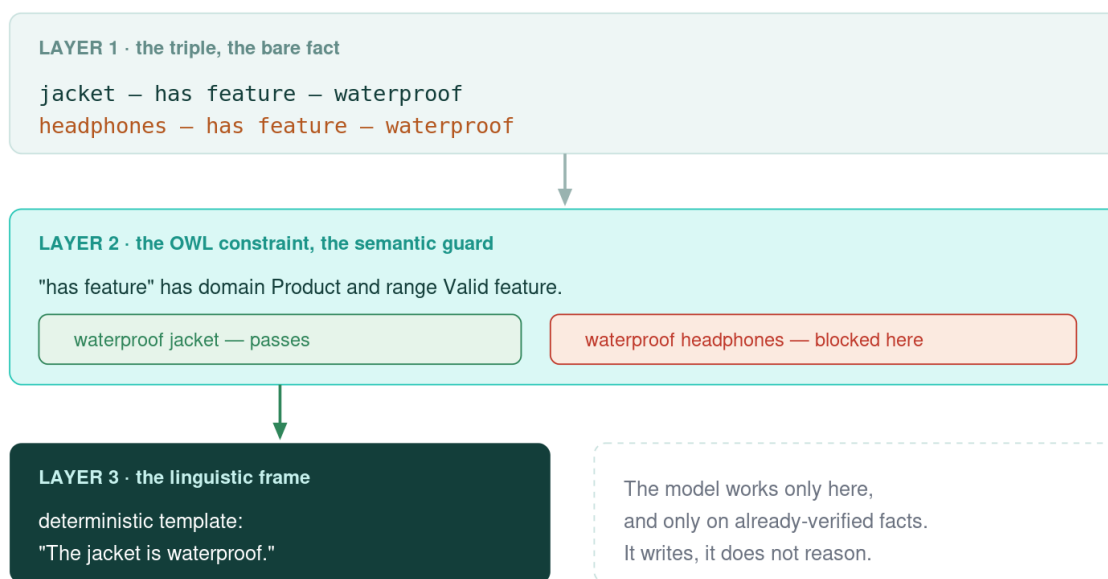


Fig. 4 – The three layers that stop hallucination.

The same mechanism, [seen from the side of annotations](#): the knowledge graph is reliable but unreadable for a non-expert, the LLM is readable but unreliable, and the move is to use the model as a translator rigidly constrained by the rules of the ontology, never as a source of truth. The truth sits in the graph, the fluency is supplied by the model, and the annotations keep the model from inventing or from dropping a piece of data along the way.

An everyday example makes the thing more concrete than any diagram. A search on a commerce site: blue waterproof jackets under two hundred euros, size M, two-day shipping. A model with fragments alone can return a fluent and wrong answer, maybe an out-of-stock product, or "waterproof" taken as a

descriptive flourish rather than as a binary feature to verify. With the ontology in front, every piece of the request resolves cleanly: blue is a value of the color attribute, waterproof is true or false, size M is a filter over the variants, two days is a constraint on shipping. The graph supplies the verified facts of right now, real availability, real price, real timing, not the statistics from when the model was trained. And the orchestration builds a precise query, waterproof feature equals true, price under two hundred, size M available, shipping within two days, stock above zero. The interpretive decision does not happen inside the model at runtime, it happens once, when you design the ontology. The same pattern scales beyond commerce, anywhere a plausible but wrong answer costs dearly: in healthcare, among symptoms, diagnoses and contraindications; in finance, among account states, rules and regulatory constraints; in internal document search, among policies, approval chains and access control.

The problem, moreover, has a longer history than the current wave of models, and knowing it helps you not to treat it as an absolute novelty. Converting structured data into natural language has been a research field for years, with dedicated benchmarks, such as the WebNLG challenge, built to measure how well a system translates RDF triples into readable prose. The first neural-network approaches were fluent but dropped specific data and struggled with entities never seen before, and ontological annotations were born to cure precisely that illness, not to make the text prettier, to keep the system from losing or inventing facts while it makes them readable.

Something needs to be said that dismantles a widespread misconception, because in many companies the magic word of the moment is GraphRAG, uttered as if it were the solution. Retrieval augmented from a graph instead of from a heap of text is a step in the right direction, it reduces hallucinations, it gives the model better material to start from. But it remains a technique, not an architecture. It does not enforce semantics, it does not validate the structural coherence of the fragments it retrieves, it does not keep unrelated nodes from being mixed into a plausible and wrong answer. With GraphRAG the model guesses better, with better material, but it keeps guessing. The difference between a system that guesses well and one that knows is entirely in where you put the validation, before the model speaks or after it has already spoken.

05 The graph is the contract

One thing strikes you when you stop looking at the individual products and look at the form. [Comparing](#) two worlds that start from opposite extremes, the open, composable stack made of open-source pieces on one side and Palantir's vertically integrated platform on the other, you discover that they converge on the same form. A typed graph at the center, with entities and relations that have a name. The indexes, the vector ones and the lexical ones, at the edges, as entry doors for finding the point to start from. An orchestration layer that treats retrieval and writing as tools available to the model. And evaluation wrapped around everything, because without measuring you do not know whether the system works or is lying to you well.

The heart of the matter sits in three words, the graph is the contract. The schema of the ontology works simultaneously on three fronts. It is a prompt, because it tells the model which entities and relations it is dealing with. It is a validator, because a write that violates the schema gets rejected. It is an API surface, because it defines what can be asked and what can be done. And models reason distinctly better against a typed graph with named relations than against raw text or against a soup of unlabeled links.

The reference architecture

The form everyone converges on: typed graph at the center, indexes at the edges, orchestration on top, evaluation around.

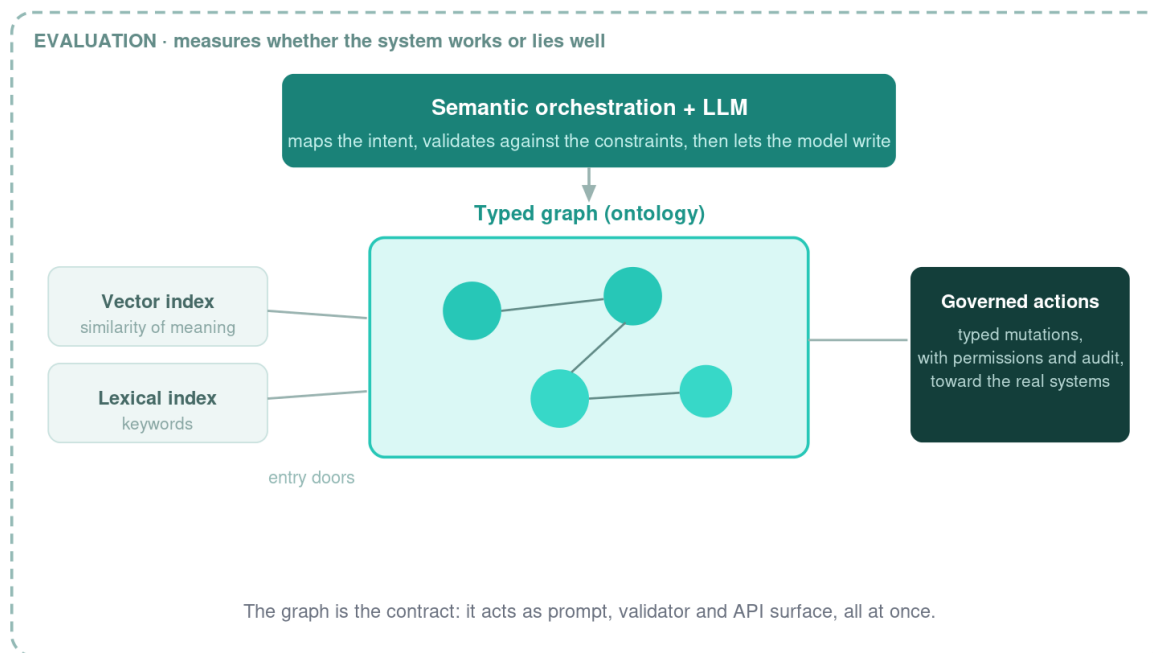


Fig. 5 – The reference architecture.

Palantir, whether or not you like it as a company, has taken this idea to the highest level of maturity around, and it is worth understanding even just to copy the approach. Its Ontology has three primitives. The object types are the schemas of the entities, with typed properties and a primary key, fed by the data pipelines. The link types are the relations, typed and directional. And the action types, the part that almost every open stack ignores, deserve a section of their own and I will come back to them shortly. Above everything run functions that execute retrieval inside the security model, and this means something not to be underestimated, a user obtains from the graph only the objects they have the right to see. The permission filter is guaranteed by construction, not stuck on top of things already built. Around the Ontology the layers stack up, the building of the pipelines that feed the graph, the logic that uses the models as tools, the low-code applications for end users, and everything is crossed by versioning, provenance tracking and a security model that propagates all the way to the model's context. The lesson is clean: on one side a composable, portable ecosystem, with better unit costs, where however you build the governance yourself; on the other an integrated, opinionated platform, where security and traceability are native but the price is a structural lock-in, because an organization that has encoded its operations into thousands of functions inside that Ontology does not migrate lightly. Neither of the two is right in absolute terms, it is a conscious choice between two models of control, to be made knowing what you gain and what you hand over.

Beneath all this a detail silently determines the quality ceiling of the system, and almost nobody looks at it: how you break documents into chunks and how you retrieve. Dense search, by similarity of meaning, and the old keyword search fail in complementary ways, so it is worth running both and fusing the results. On top you put a finer reranking with a model that compares question and candidate in pairs. And the quality of the whole rests on how you cut the texts to begin with, because badly done chunking sets a low ceiling on everything that comes after, however sophisticated the model. The real test of a vector database is the strength of permission-filtered retrieval, while the numbers of unfiltered search are marketing.

Above the graph sits the bridge, the semantic orchestration layer, the piece talked about least that does the most work. It receives the user's intent, maps it onto the concepts of the ontology, chooses the right entities and relations, builds the query, validates the data against the constraints before the model sees it, and only then lets the model write. The model's job shrinks to writing, and the interpretive decision, the dangerous one, shifts to the moment you design the ontology, once, done well, instead of repeating itself with every single answer inside the black box.

A further layer the sharpest CIOs already have on their agenda, and it is context. In a world where the base models converge and increasingly resemble one another, [the defensible moat](#) sits in the data orchestration layer, and the applications that run on a model without that layer have no defensibility at all. There are four streams of context to orchestrate into a single payload, the user's intent in the current session, their long-term history, the system context with its guardrails, and external data in real time. The engineering challenge is deciding what deserves a place in the context window and what should be discarded, even before concatenating the sources, because every token has a direct cost and at scale that cost eats the margins. Over histories spanning months or years the solution is a graph memory structure, that decides in real time what to surface, what to compress, what to forget entirely.

The four context streams

Every token has a cost. The orchestration decides what to surface, what to compress, what to forget.

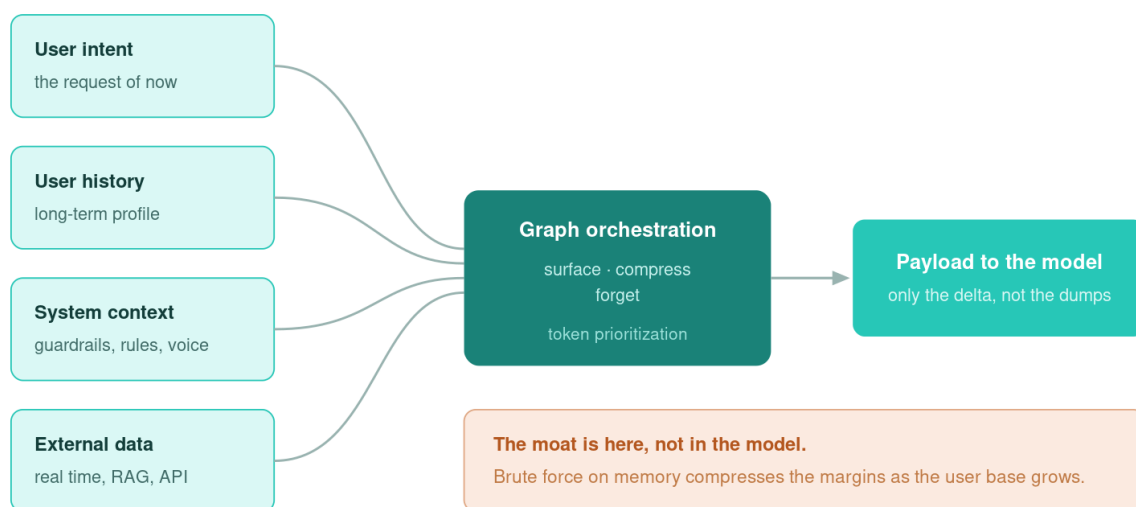


Fig. 6 – The four context streams and token prioritization.

Two real failures are worth more than any principle, because they show where the money ends up. A customer-success assistant at a software company, with a naive approach, pulled fifty thousand words of raw email into the context window, saturated the limit, cut precisely the most recent and critical emails, and missed a contract renewal, for having prioritized volume over relevance. The engineered version passed to the model only the dense summaries of the old logs plus the full text of the three most recent interactions. A financial robo-advisor, on every page refresh, replayed the entire portfolio history, twenty holdings, five news items, with unsustainable API costs because ninety percent of those tokens were static, redundant data; the fix was architectural, cache what does not change and inject only the delta, the values that actually changed beyond a threshold. Every input and output token has a direct cost, and the brute-force approach to memory compresses the margins as the user base grows. Your context

architecture directly impacts the income statement, and that is a sentence a CFO understands better than any diagram.

The last layer is the one that turns a model into an agent. [An equation](#) sums it up well and is worth keeping in mind, AI equals LLM plus harness. The harness is everything around the model, the loop, the tools, the memory, the guardrails, the repeated retries, the traceability. The model generates, the harness decides whether to trust it. The model is the small part, you swap it next month for a better one, while the harness is what you actually build and maintain, and that is where reliability is decided. A search engine can tell you what the rule is now, a harness can tell you what has changed since last time, and in a company the second question is worth much more than the first.

Inside a harness the engineering effort should be spent almost entirely outside the model. The loop is observe, reason, act: search for updated official sources, classify them and extract the rules with numbers and dates, write an output usable by a human. After reasoning the agent checks itself, does it have enough sources, are they reliable enough, and if that is not enough it rewrites the query toward better sources and repeats, up to a cap on retries. Around the loop live the interchangeable search tools with automatic fallback, the JSON schema validation that retries with a corrective prompt if the output is truncated, the memory that compares the new result with the last saved one and shows what has changed without creating duplicates. Always the same principle, at every level of this discussion: durable value lives in the system that decides when to trust the intelligent component, more than in the component itself, and that system you build and maintain, while the model underneath you can swap out tomorrow.

06 Governed actions

I pick the Palantir action types back up, because this is where the conversation about ontologies and AI leaves document search and enters real operations. Almost every RAG system around knows how to retrieve information. Then, when it needs to write something, change a state, make something happen in the world, it relies on improvised shortcuts, direct calls to the database, hand-stitched scripts. Actions are the frontier that pure retrieval ignores.

An action type is a declared mutation, typed, with verified permissions, validation rules, explicit side effects, an audit trail, and the ability to require a human's approval before it executes. The ontology becomes readable and writable, but only through actions, never through a direct handle on the database. I want the principle written into every project: define a closed vocabulary of possible mutations, with their preconditions, and give the model that, never free access to write whatever it wants.

From the harness side the same idea dresses up as operational governance. The guardrails are not decorative, they can really fail and stop the process. An inbound check blocks suspicious requests before any network call. An outbound check verifies that the output carries a source, a confidence level, a disclaimer, and flags as an alarm, not as a silent green light, the absence of official sources. One episode is worth more than a theory: on a topic of visa rules an agent had obtained eight sources, none official, the check correctly turned yellow, the agent rewrote the search on its own toward government sources and found seven of eight official. In that moment it stopped looking like a script.

Grounding against hallucinations meets governance and the human in the loop here, on the terrain of actions, and it is no coincidence. Retrieving wrong information costs an imprecise answer. Executing a wrong action on a real asset costs far more, and the difference is exactly the layer you decide to build between the model and the world. An ontology that describes only how the domain is made is half the

work. The other half describes what it is permissible to do with it, by whom, with which controls, and leaves a trace of everything done.

The nature of the ontology changes this way too, from photograph to operational tool. In Palantir's model the graph is writable as well as readable, and the writes can flow all the way back to the source systems, but always and only by passing through actions, never with a direct, opaque write. Every change carries with it who made it, when, with what validation, and stays reconstructable thanks to provenance tracking. It is the difference between an AI that tells you what you should do and an AI that does, inside a fence of rules someone wrote and someone is accountable for. The more companies connect agents to these graphs, the more the fence matters, because an agent that can only read gets an answer wrong, an agent that can act without a fence gets an operation wrong, and the second has a cost measured in real money, not in embarrassment.

07 Fraud as a network

So far the reasoning has been architectural, and it risks staying abstract without a problem a CEO recognizes at a glance. Fraud detection is the right one, because it shows everything at once, why structure matters, why semantics matters, and why speed of execution is not enough without meaning.

[The starting point](#) flips the classic way of looking at the problem. Modern fraud is not a transaction problem, it is a network problem. A single operation, looked at on its own, seems legitimate almost always. It is the pattern of relations around that operation that tells the real story: dozens of accounts linked to the same device, the same phone number, the same legal person, in configurations that no transaction-by-transaction control can see. A graph that merely stores that device A is linked to account B is not enough, because it does not distinguish a family sharing a laptop from a fraud ring using a single device for hundreds of fake accounts. You need the semantics, knowing what that node means, what that edge means, which constraints hold, how risk propagates across different entity types.

Here the ontology enters in a tangible way. It defines the entity types, the legal entity, the authorized representative, the device, the IP address, the transaction, the account. It defines the relations, controls, is registered with, is associated with, originates from. It defines the properties, the jurisdiction code, the device fingerprint, the risk score, the timestamp. And it defines the constraints, the suspicion rule, a device associated with more than five distinct legal entities in twenty-four hours goes to review. That rule lives in the ontology, written once, not wired into the code of twenty different pipelines. Humans define the ontology, the graph database executes it automatically: the design is weeks of human work, the execution is at machine time, without intervention, on every payment, in milliseconds, before the transaction completes.

The runtime flow, for a cross-border payment, reads as three chained checks. First the device, is it associated with other accounts and what risk score do these have. Then the legal entity, is the jurisdiction sanctioned, is the representative on a watchlist. Then the transaction pattern, does the combination of device, IP, entity and amount resemble a fraud signature already seen. Three steps, a semantic graph that crosses them knowing what it is crossing, an answer before the money moves. The numbers that accompany these systems, when they work, are significant: a global bank that cuts false positives by over forty percent in trade surveillance, a credit operator that goes from queries above three seconds, with thirty percent of high-risk queries missed, to under eight milliseconds after rebuilding everything on a graph. One honest caveat I owe, though: figures like these almost always come from the people selling

those platforms, with no public baseline and no cost side, and they should be read as indications of direction, not as proof. The argument of this article does not rest on them, it rests on the logic of the graph, and the numbers serve only to give an order of magnitude to what that logic makes possible.

The further value is organizational before it is technical. In a company that manages risk, the KYC team, the people monitoring transactions and the ones investigating fraud look at the same data and each define "risk" in a slightly different way. The ontology becomes the shared semantic layer that aligns the whole organization on a single definition, and fraud detection moves from the reactive hunt for the rule that tripped the alarm to a proactive intelligence about relations. Without the ontology every investigator stitches the pieces back together by hand. With the ontology the stitching is done by the system, and it does it the same way every time.

08 The standard and the extension

The practical question arrives, right after understanding why you need it: do we build this ontology ourselves. The first answer is counterintuitive for anyone with an engineering culture of doing it in-house. In many sectors the hardest part has already been done by someone else, often decades ago, with years of committee work, formal governance and mappings to every other standard in the sector.

[Rebuilding from scratch](#) concepts like "patient" or "diagnosis" is rework disguised as craftsmanship, and the clean in-house schemas are the way you end up privately reinventing SNOMED, one node label at a time. It is worth knowing what is already on the shelf, because there is a lot of it and almost nobody in a company knows it.

In healthcare and pharma the shelf is crowded. SNOMED CT is the clinical terminology, hundreds of thousands of concepts, used in more than fifty countries. ICD-11 is the World Health Organization's classification of diseases. MedDRA covers adverse events and is owned by the regulators themselves. RxNorm normalizes drug names. UMLS acts as a Rosetta Stone and maps one onto another. The Gene Ontology describes the function of genes with over a million annotations backed by experimental evidence. In finance there is FIBO, the Financial Industry Business Ontology, built by the EDM Council and OMG, with thousands of classes in production. In the supply chain there is no single ontology but a stack, GS1 with EPCIS for events, SCOR for processes, the W3C PROV standard for provenance. In law there is LKIF. And as general glue there are schema.org, which you use every time you optimize a site for search engines without knowing it, and Wikidata.

Sector	Standard	What it covers	Who maintains it
Healthcare & pharma	SNOMED CT, ICD-11, MedDRA, RxNorm, UMLS, Gene Ontology	clinical terminology, diseases, adverse events, drugs, genes	SNOMED Intl, WHO, NLM, academia
Finance	FIBO	entities, instruments, securities, derivatives	EDM Council, OMG
Supply chain	GS1 EPCIS, SCOR, W3C PROV	events, processes, provenance	GS1, ASCM, W3C
Legal	LKIF	obligations, permissions, prohibitions	academic consortium
General	schema.org, Wikidata	people, organizations, places, products	web consortium, community

Table 1 – Standard ontologies by sector.

FIBO, in finance, counts thousands of classes in production, from fundamentals to business entities to financial instruments to derivatives, and this very richness triggers a mistake I often see made, mistaking size for a reason to build in-house. It is not. Forty curated classes, actually maintained, aligned to your codes, beat two thousand classes downloaded and never reviewed, because size counts for very little, and what gives value is that the borrowed piece already agrees with everyone else's. In healthcare it is even clearer: UMLS maps SNOMED, ICD, RxNorm, MedDRA to one another, and mappings like those exist precisely because the concepts of two standards never align one to one.

The thing none of these standards gives you for free, and which is the real work, is the care. Reuse is not free, it trades modeling work for understanding work, and it is usually worth it. There are three traps. The first, do not import the whole standard, take the slice you need and maintain it, because a curated slice beats the full import only if you actually maintain it. The second, mapping is where the real work hides, because the concepts of two standards almost never align one to one, and a subtly wrong mapping is worse than no mapping. The third, the fit is never perfect, because the standard embodies someone else's view, and extending it without breaking it requires more understanding, not less.

There is a deep reason why these standards are safe to build on, and it is who maintains them. They do not come out of a vendor, they are kept alive by the World Health Organization, consortia of regulators, national libraries of medicine, non-profits and academic consortia on public funding. It is slow, governed infrastructure, boring in the best sense of the term, that survives any product roadmap and any acquisition. And there is a bet about the near future: as you connect agents to graphs, the value of a shared, machine-readable vocabulary goes up, not down. An agent reasoning over data grounded in a standard like FIBO stands on thousands of classes of settled agreement about the meaning of words. An agent reasoning over a schema invented last sprint stands on the intuition of the moment, and when it gets things wrong you have nothing to appeal to. The ontology is what keeps a knowledge graph from sliding into a big heap of duplicated noise with edges.

The strategic criterion that comes out of this I keep as a compass: the standard makes you interoperable, your extensions make you interesting. Half the value of a corporate knowledge graph can sit precisely in the part nobody has standardized, the part not even the competitors have modeled, because that is where your enterprise's specific knowledge lives. You reuse where the domain is stable and regulated, you build where you are genuinely original, and at every moment you know which of the two feet you are standing

on.

Reuse, build, or hybrid

The standard makes you interoperable, your extensions make you interesting. Almost always the answer is hybrid.

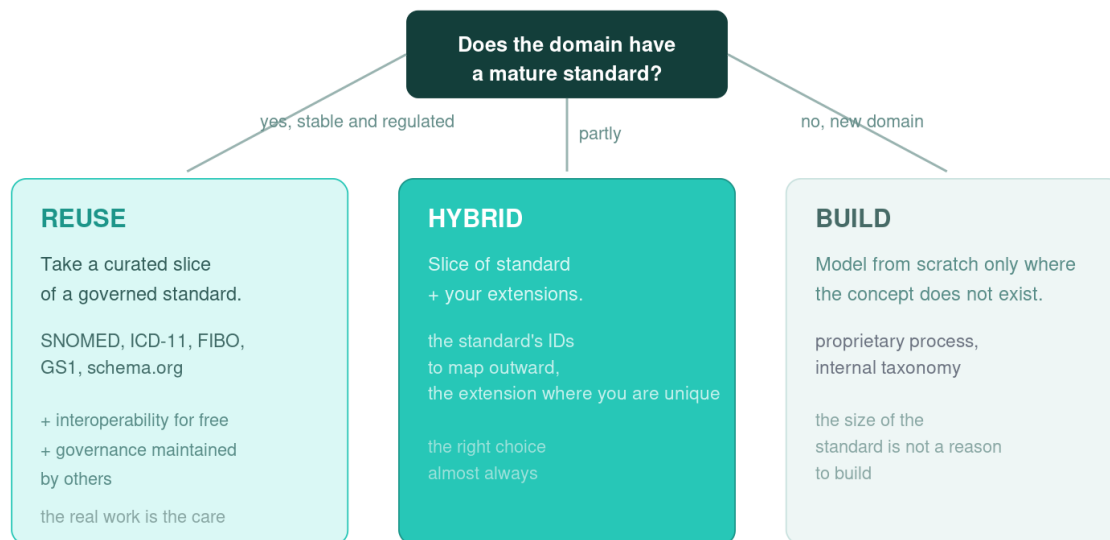


Fig. 7 – Reuse, build, or hybrid.

There remains the problem of how you start, more psychological than technical. They call it [the blank canvas problem](#): too many valid ways to model a domain and no anchor point, so the debate over what our entities are goes round in circles for days. One anchor is called POLE+O. POLE is a historical model from intelligence analysis and law enforcement, four types with which heterogeneous data has been connected across different agencies for decades, Person, Organization, Location, Event. The plus one is Object, the fifth type, the container for everything that does not fit the first four: if an entity is not a person, an organization, a location or an event, it is an Object. Five base types to start with immediately. The power does not lie in the five types, it lies in the fact that on top of the base type you put the domain type, a patient is Person plus Patient, a sprint is Event plus Sprint. You query at the base level and pick up diagnoses, incidents and transactions together, and you automatically unify different sources because a Person from HR and one from the ERP are already in the same container. A graph you can query in week one teaches you more about your domain than a schema you debated for a month. With the honesty, declared, that POLE is a starting point and not a complete ontology, and that in abstract domains, financial instruments, business rules, the Object that holds everything risks holding nothing.

POLE+O, five types to start right away

POLE (Person, Organization, Location, Event) plus O (Object). On top of the base type you put the domain type.

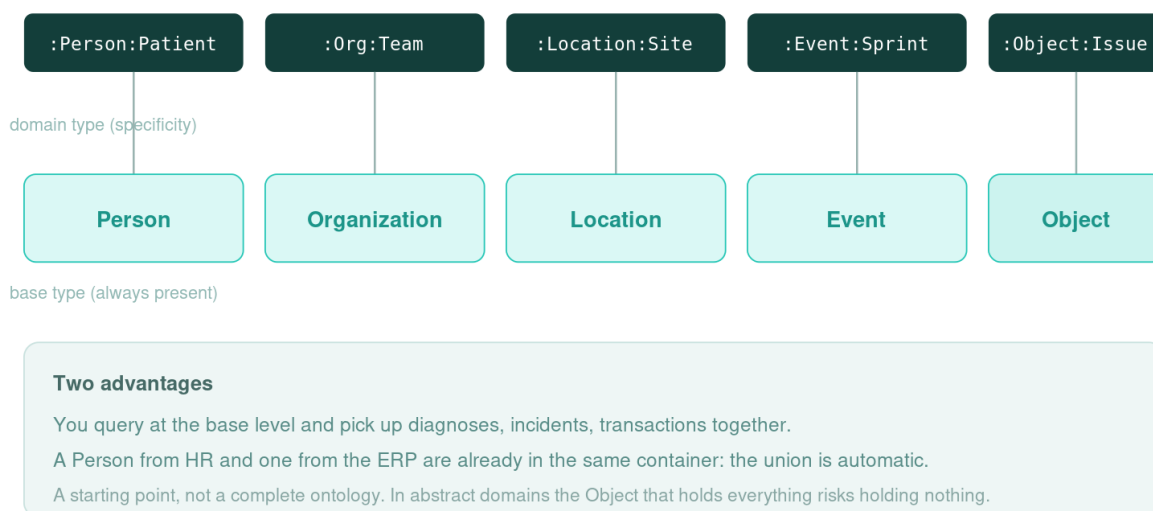


Fig. 8 – POLE+O: four base types plus Object.

09 Extraction and population

The economics of these projects has changed, because building a graph, the part that until yesterday cost months and specialists, has become cheap. Recent techniques have cut construction costs by orders of magnitude, and in some benchmarks the precision with which a model extracts entities and relations from a text approaches ninety-nine percent when you give it a reference schema to stick to, against much lower numbers when you leave it unbridled.

On that number, though, you have to keep a cool head, because even very high precision does not tell you how much [the missing part](#) weighs, and that part almost always gathers the distinctions that matter most, the edge cases, the exceptions, the boundaries of responsibility, not the noise distributed at random that you would be tempted to imagine. An ontology extracted at ninety or ninety-five percent is perfectly fine for search, for BI, for a RAG, and it turns fragile where the margin of error is not admitted, compliance, safety, engineering, product configuration. The real problem is not accuracy, it is the alignment of meaning among all those who will use that model.

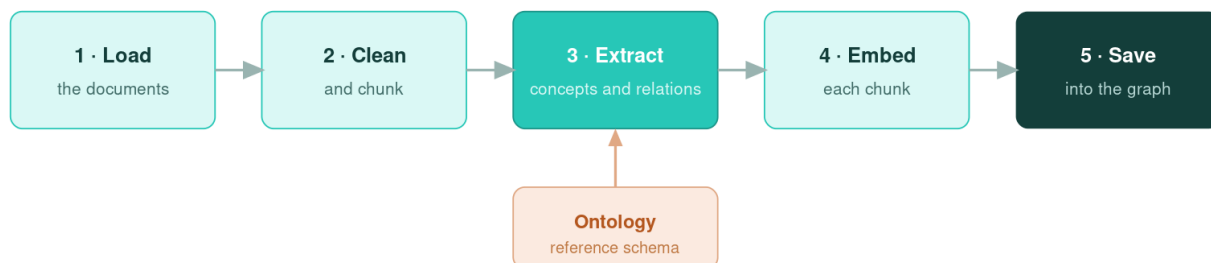
That detail, the schema that guides extraction, is the keystone and it returns in every section of this discussion, it is the same slice of curated standard from before, used here as a rail for extraction.

[An end-to-end pipeline](#) can be built entirely with open-source tools, runs on a laptop and requires no licenses. The sequence is clean: you load the documents, clean them and break them into chunks, extract from each chunk a small graph of concepts by passing the extractor the ontology as an optional blueprint, compute the embeddings, save everything to a graph database linking each chunk to the document and to the extracted subgraph. The piece I set aside concerns the queries, because there are five different ways to query the same graph, from enriched similarity search to the automatic generation of structured queries, and the lesson is that there is no single right strategy, you choose by type of question, balancing cost, latency and quality. The more precise and descriptive the ontology, the easier it is for the model to

write the right query, and schema governance directly improves self-service accuracy.

The build pipeline with models

From documents to a queryable graph. The ontology passed as a blueprint guides extraction and raises precision.



With the schema guiding extraction, precision reaches close to 99%. Without it, the model goes by feel.

Fig. 9 – The construction pipeline with the models.

Those strategies, up close, show that there is no single path. Enriched similarity retrieval fetches the chunks closest to the question and completes them with the contiguous ones, good when you need detail and citation of the source. Community summaries group the graph into clusters of related nodes and summarize each cluster, good for synthesis questions that cut across many sources. Direct generation of structured queries passes the model the graph schema and lets it write the query, excellent for questions with a precise answer when the schema is well defined. And then the combinations, with a detail that matters in production, the fallback: if query generation fails, the system falls back to similarity retrieval instead of returning an error. The lesson is one of portfolio more than of technology, you choose by type of question weighing together the token cost, the latency, the quality, because a perfect answer that costs ten times as much and arrives in ten seconds, at scale, is not sustainable.

On how to extract, the debate is real, and [mapping it honestly](#) helps. The options are a large model used as is, a smaller model trained on your domain, a classic rule-based pipeline, or a combination. The large model starts right away but is costly and variable. The trained small model costs upfront and is then cheap and consistent. The classic rules are deterministic and verifiable but fragile out of domain. No absolute winner, an architectural decision that depends on your constraints, and not a technical detail to delegate, because how you extract shapes the quality, coherence and usefulness of the entire graph.

Under the choice lies an axis worth making explicit, between top-down and bottom-up. Top-down, a schema or an ontology guides extraction and the data is bent to that structure, and it is exactly the schema guidance that brings precision close to ninety-nine percent. Bottom-up, the schema emerges from the data. The classic rule-based pipelines have a dignity that the enthusiasm for models makes people forget, the linguistic patterns that recognize "is a" relations have existed for decades, they are deterministic and verifiable, and in a stable domain they work without surprises. The most promising approaches today are hybrid, where explicit ontological rules constrain a trained model to be aware of them, so that the reasoning stays anchored to the symbolic constraints instead of drifting away from them at will. The rule I draw from this, and that ties this section to all the others, is that giving the model a reference schema is the multiplier that raises quality at every stage, from extraction to query.

One case struck me for its practical elegance. [A tool](#) takes a website or a folder of documents and induces the ontology from the documents themselves, with no schema fixed in advance, assigning to every attribute and every relation a confidence score. The interesting part is that the schema self-corrects: having found orphan nodes, the tool asks the model whether the problem is the schema, the model proposes new properties, the pipeline validates them and redoes the extraction with the widened schema. The graph grew its own schema to fit the data, once, then it settled. And every statement stays traceable back to the page of origin, with a fingerprint that says exactly what changed between one crawl and the next. The difference between having scraped a site once and having a verifiable snapshot. The graph is the artifact, the answer to a question is only the first thing I asked of it.

The numbers of that case give the measure. From a corpus of pages of a news site came a graph of 779 nodes and 1131 edges, with the types induced from the texts, article, publisher, company, topic, person, exchange, index, and the relations among them, published by, is about, mentions, listed on. The confidence is differentiated honestly, the Nvidia ticker extracted from a passing aside weighs 0.6, the semiconductor industry asserted across many sources weighs 0.8. And a question like what is trending translates into a concrete and verifiable operation, counting the "is about" edges per topic, with results inspectable one by one, eighty-seven articles on AI, seventy on quantum, and so on. The opposite of the plausible answer taken on trust: every row traces back to a file, every file to a page with its fingerprint.

From here comes structured memory too, which matters for those who build agents. It is organized on three levels above the five-type model, a short-term memory for the session, a long-term one made of persistent relations among entities, and a reasoning memory that records the traces of decisions, the thought, the action, the observation, as linked nodes. A chain of reasoning is not something you traverse in a store of vectors, you traverse it in a graph. It is the difference between an agent that remembers what it decided and why, and one that starts over from scratch every time, faking memory.

10 Semantics outside the graph

Now the countercurrent, the section most useful to those who hold the purse strings because it saves money and time, and which I also need in order to dissolve a misunderstanding I may have fed myself so far, because when I say knowledge graph I mean a structured, governed semantic layer of which the typed graph is the most common seat but not the only one, given that the same ontology can live inside a graph database, inside an external rules engine, or in annotations that constrain the model, and what counts is the semantic contract, much less the container in which you write it. In every meeting, meanwhile, an automatic line of reasoning goes around that sounds like this, we need an ontology, ontologies are written in RDF, RDF is a graph, therefore we buy a graph database, and [every single link](#) in that chain is weaker than it seems.

Graph databases are neither new nor unique. The network model treated relations as first-class citizens as far back as the late sixties, and relational databases too, by definition, are relational. The real strength of graph databases is narrow and real, recursive queries, traversals of unknown depth, when you have to trace back a chain of suppliers of suppliers or a network path of unpredictable length. Outside that the advantage thins out fast.

One distinction clears up the misunderstanding: when we say relations matter, we are saying three different things. Existence can matter, the very fact that A is connected to B is already information, and in certain domains, intelligence for example, it is the heart of the work. Traversal can matter, can I get from A

to C. And semantics can matter, what that edge means, is it transitive, is it symmetric, is it functional. Graph databases handle existence and traversal beautifully, they do not handle semantics, which is the domain of logic and not of graph theory. Confusing the three meanings is the way you buy a tool for a problem you do not have.

There are then practical limits that sellers rarely mention. The most expressive variant of OWL is undecidable and almost nobody implements it outside academia, the tractable one is costly, and general-purpose graph databases do not perform inference on their own, they need external reasoning engines. In pharma, SNOMED CT uses a graph database as a store, but the logical classification has to be run outside the database, in batch, with a dedicated reasoner, which complicates the pipeline. Relations are limited to two ends at a time, and as soon as you need more, an event with a time, a place, actors and an amount, you have to resort to hypergraphs or to rounds of reification that weigh the model down. And the security model is often underdeveloped, a path from A to C that passes through D can reveal information the user should not see, and handling this role-dependent non-monotonicity is anything but trivial. None of this says graphs are useless, it only says they should be chosen for the right problem, with your eyes open to the work that stays outside the database.

The distinction that clears everything up is between facts and rules. In description logic one speaks of the A-Box, the assertions, the facts, what a graph database stores beautifully, and of the T-Box, the terminology and the rules, the inference, what a graph database does not do. The graph query languages are excellent at traversing, they do not reason, and for inference you need external engines. One sentence is worth an entire consultancy: a graph database without an ontology is just a less convenient relational database. What makes the graph intelligent is the semantics you put on top, never the graph in itself, and that semantics lives in logic, not in graph theory.

The pragmatic proposal is a two-step path that starts from where the company already is. Step one, do not build an ontology, expose to the models the rules you already have, because almost every structured enterprise has rules engines, encoded policies, business logic already written in systems nobody looks at anymore, and making them accessible to a model through a standard protocol like MCP is often the right first step. Step two, if you need more, logic programming, with languages like Datalog that guarantee decidability, are fast, native to facts and rules, and moreover readable by business stakeholders, which cannot be said of RDF and OWL. The moral is simple, and graph databases remain useful, they simply should not be the default choice, and certainly not the first. The right question is where do I put the semantics, not whether or not to buy a graph: inside the graph, in an external reasoner, or in annotations that constrain the model. These are three seats of the same thing, and everything I have written about the ontology as a contract holds identically in each, only where the contract is written and who enforces it changes. Answering this question well, before signing any license, is half the saving.

Where semantics lives

The same semantic contract can sit in three seats. What changes is where it is written and who enforces it, not its nature.

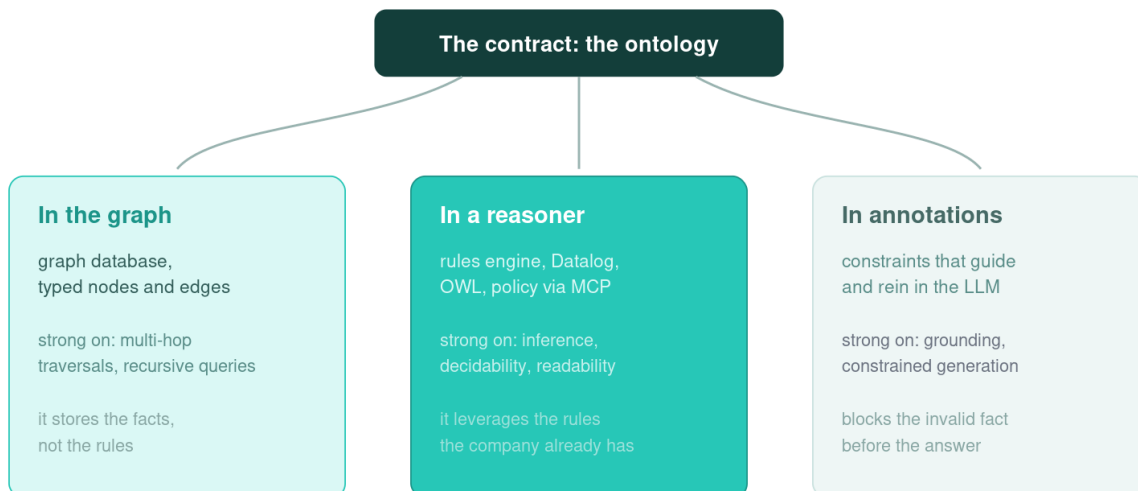


Fig. 10 – Where semantics lives.

11 Ontology drift

So far we have talked about building. Almost nobody talks about what happens the day after, when the ontology that governs the graph slowly begins to no longer correspond to reality. They call it [ontology drift](#), and the starting observation should make every sponsor of these projects think. The technology is mature, the construction costs have collapsed, and yet adoption in production is flat: according to an industry survey, a little over a quarter of enterprise knowledge graphs actually reach production, year after year, almost without moving. If building has become easy and the graphs in production do not increase, the bottleneck is no longer building. It is keeping the meaning aligned while the business changes.

Semantic models do not fail all at once, they diverge. A definition changes in one department, finance redefines what counts as an active contract by excluding early renewals, and nobody updates the model. A schema evolves on its own, a new table, a renamed column, a refactored pipeline, and the ontology knows nothing about it. The same logic gets duplicated in five places, a SQL view, the BI layer, the application code, the AI prompt, and it gets updated in three of those five. No system truly owns the meaning, the logic scatters, there is no longer an authoritative source, and the teams freeze the definitions for fear of breaking something downstream. Semantic debt accumulates in silence, like the [cognitive debt](#) I have been talking about for a while, the hidden cost of having delegated without structure, which you do not see while it grows and which presents you the bill when you least expect it.

One detail makes the subject urgent right now, and it concerns the difference between the old BI and the new AI. A dashboard with slightly dated definitions limps, shows a slightly skewed trend, and the human analyst reading it notices, adjusts, corrects by feel. An agent does not. It treats the ontology as reference truth and confidently returns an answer built on now-outdated logic, and nobody notices until the damage is done. Semantic divergence kills AI adoption faster than it kills BI, because AI does not have the critical sense of the analyst who compensates downstream. It fails in silence, and the silence is the dangerous part.

Ontology drift

The semantic model does not fail all at once, it diverges. And AI, unlike BI, errs in silence.

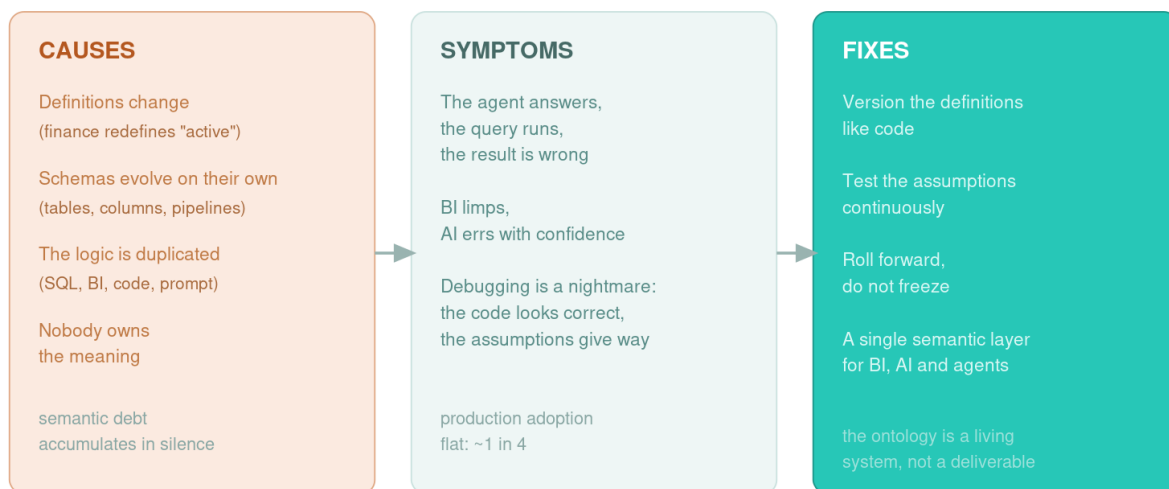


Fig. 11 – Ontology drift: causes, symptoms, remedies.

And when it fails it is hard even to notice, because everything seems to work. The agent answers, the query runs without errors, the result is formally valid, only it no longer corresponds to what the business means. Debugging becomes a nightmare, because the code looks correct, and indeed it is, what gives way are the assumptions underneath, which no longer hold. A small change in a definition, "active contract" that from today excludes early renewals, propagates like an invisible error into every answer built on that word, and nobody sees it until someone, by chance, compares a number with reality and finds it does not add up. It is the reason projects that looked successful stall after go-live: the problem has shifted from the technology to the operating model, and that is where the investment must shift, from building once to maintaining always.

The teams that do not fall into the trap make a few concrete and documented moves. They treat changes of meaning as changes of code, versioned, reviewed, tested. They explicitly version the business definitions. They continuously test the semantic assumptions, with automatic checks on the constraints and the relations. When a definition changes they do not freeze, they manage the transition with deprecation windows and migration paths. And they use a single semantic layer shared across BI, AI and agents, so that inconsistencies surface right away instead of nesting. The ontology should be treated as a living system, not as a project deliverable you hand over and file away. It evolves because it is connected, not isolated.

The drift is hard to stop for reasons more organizational than technical. Ontologies get treated as static artifacts, with effort concentrated at the start and then zero maintenance, in pure project logic. The data stack and the ontology live in separate worlds, so the semantic layer does not know that the data has migrated to a new warehouse. The expertise to touch it is rare and costly, and this isolates the layer within a small group of specialists that almost nobody consults, with the paradox that the fewer people look at it, the less the drift is seen, the less pressure there is to maintain it. And every update risks breaking something downstream, a query that stops running, a dashboard with different numbers, so the natural incentive becomes to freeze everything, and freezing is exactly what produces the divergence.

The same gap has another face, the distance between those who build the data and those who use it. The data warehouses are built by engineers, the dashboards are built by analysts, and in between a chasm opens up, the semantic layer, almost always made by hand, poorly documented, [perennially behind](#). To build that layer an engineer has to understand every foreign key, every chain of relations, every edge case where a join changes nature, then interview the stakeholders for the business questions, then design the wide tables that flatten a complex relational model into something an analytics tool can query without six joins. Days or weeks, for a schema of medium complexity, and a job that the moment after already begins to age. The direction is to generate this layer in an assisted way, compressing weeks into hours, but the point stays identical regardless of the tool: that bridge between data and meaning is where the costs are born, and treating it as a one-off job is the guarantee that it becomes debt.

The same intuition sits behind the [Open Knowledge Format](#), the open standard proposed by Google, summed up with an effective analogy, a Git for knowledge. As Git standardized versioning, portability and collaboration on code, the idea is to do the same with organizational knowledge, today fragmented across wikis, document managers, data catalogs, repositories, and above all in the heads of the most expert people. The bottleneck of AI has shifted from the model's intelligence to context, and context is organized, versionable, portable knowledge, which today almost nobody manages the way they manage their own code.

12 Governing meaning

Taken seriously, the word ontology carries you far from the choice of a product or a database. It carries you into the boardroom, because the real decisions concern who owns the definitions, more than the technology, and the definitions constrain everything downstream. [Three questions](#) deserve a place on the board's agenda, and every CEO and every CIO should be able to answer them.

The first, who owns the chart of entities. Master data management is the closest candidate, but it rarely works at the level of definitions and relations; the chief data officer is usually the right seat, but their mandate often does not yet cover this governance of structure. The second, treat the definitions of the core entities as decisions of relevance to the board, but only those that touch revenue, risk, regulatory exposure, customer trust, delegating the rest. The third is the most uncomfortable: what does your AI strategy assume about structure. Almost every AI strategy assumes good data, almost none assumes structured data, and that gap is exactly the point where artificial intelligence programs run aground after launch.

The board does not need to put its hands on every definition, and the effort must be calibrated. Only the entities that touch revenue, risk, regulatory exposure, customer trust deserve the board's attention. The rest is delegated. It is a minimal but beloved governance, concentrated where the consequences are real and light where they are not, the only way for the governance of structure not to become itself a bureaucratic bottleneck. The definition of "customer" you choose decides who enters the retention numbers, who falls under a regulatory obligation, who an AI system will treat as the target of a decision. Once written into the graph, it stops being implicit and local and becomes explicit, queryable, binding on everything downstream. A choice with an owner, a history and consequences, and treating it as a technical detail to delegate to whoever configures the database is the most common way to discover too late that you delegated a strategic decision.

There is, however, a mirror-image risk, and I take it because it is the deepest and least-spoken part of the whole story, what you might call [semantic imperialism](#). When a company declares that its ontology represents reality, the danger becomes epistemological and political before it is technical, hidden assumptions, institutional bias, points of view crushed into a single category, forced interoperability. And it does not emerge when the technology fails, it emerges when it succeeds, while the ontological commitments it encoded stay invisible. No ontology is reality. It is the official account of what an organization knows about itself, an account, not the territory. And that is precisely its value for those who must answer for decisions, because a probability is not governable, an inspectable model is. Governance needs an object, and the ontology is that object.

Here I have to untie a knot, because earlier I called for a single source of semantic truth and now I say the universal ontology does not exist, and it looks like a contradiction that dissolves the moment you distinguish two planes. Inside a company you need a single definition of "customer" around which to align marketing, finance and support, and that internal coherence is governable because it is the account the enterprise gives of itself, whereas between companies, between sectors, before reality, the same definition has no right whatsoever to call itself true, and claiming so is semantic imperialism. The source of truth you need, in short, is local and declared and not cosmic, and the mistake to avoid is not in having a shared in-house definition, it is in mistaking it for the structure of the world and imposing it on those who have good reasons to see it differently.

The problem explodes right now for a reason that sits in the very collapse of costs I hailed earlier as good news. As long as building an ontology required specialists and months, semantic models were few and precious, and scarcity kept them under control. Now that generating one costs almost nothing, [every dataset, every platform, every agent, every department](#) can produce its own structure, and the risk becomes the proliferation of many internally coherent and mutually misaligned models, anything but convergence toward a common language. The fragmentation of meaning is harder to govern than its absence, because a company without structure knows it does not have it, while a company with ten structures that do not talk to one another is convinced it has one.

The answer, then, is not to build the single, definitive ontology, which does not exist and cannot exist, it is to govern the plurality. Multiple representations, legitimate, sometimes incompatible, made visible, traceable, navigable. One sentence I keep to close the reasoning: the future belongs less to universal ontologies than to architectures capable of navigating ontological plurality without collapsing into incoherence. Not to eliminate Babel, to learn to inhabit it. On the operational plane this translates into three levels of maturity that companies systematically underestimate, the technical interoperability that makes systems talk, the document intelligence that reduces cognitive load with LLMs and retrieval, and the collaborative governance of knowledge, the human rituals with which meaning gets negotiated, conflicts get resolved, learning happens together. Organizations invest in the first two levels taking for granted that the third emerges on its own. It does not emerge on its own, and without the third the first two generate complexity without resilience.

The core that holds up everything else is that [semantics is not a property of the data](#). An LLM is probabilistic, it predicts sequences of words. An ontology is deterministic, it imposes binary logical constraints. Meaning is a different thing again, it emerges from the interaction among people, tools and context, and it resides neither in the tokens nor in the constraints. When an unforeseen failure happens in a network of actors, the way out is not found by the algorithm, it is found by the team that communicates, mobilizes heterogeneous knowledge and improvises. The real risk is misplaced trust that disarms our collective capacity to make sense, more than an AI too powerful. The machine executes correlations, the

human takes on responsibility, and confusing the two planes is the most elegant way to lose control without noticing.

One further warning would deserve a reflection of its own, what I call the cognitive Taylorism of AI. Yesterday bodies were mechanized by breaking work into elementary gestures, today knowledge is being mechanized by breaking collaboration into automatable flows, and the human in the loop often becomes cosmetic, a stamp on a decision the machine has already made. Inject a rigid ontology into a probabilistic, multi-actor system and you get an answer clothed in formal confidence, wrapped in the official vocabulary, but disconnected from the real practices distributed among people. A double risk, the false confidence and the making-invisible of semantic conflicts, because the weak signals of an interpretive disagreement get smoothed into a uniform answer. Augmented retrieval too captures only a snapshot, and the documentation is always partial and dated. Machines process signals, [humans interpret signs](#), and not confusing the map with the territory, in this field, counts as a survival rule. To govern does not mean to eliminate the differences, it means to make them visible and navigable.

A final layer concerns who holds the knowledge. A knowledge graph is the encoding of an enterprise's proprietary knowledge, its semantic skeleton, and this makes it a matter of sovereignty before it is one of technology. A company that entrusts its own meaning to a layer governed by others, inside a platform it does not control, is making a strategic decision disguised as a technical choice. The cognitive enterprise, the one that learns and reasons over its own data, needs to know where its ontology lives, who can read it, who can change it. For an Italian and European company the regulatory frame makes everything less theoretical: the [AI Act](#) demands systems whose behavior can be explained, and for high-risk cases that explainability becomes a condition to operate, not an optional to grant at the end of a project. An inspectable model is more defensible than a probability, and reconstructing the chain of relations that led to a decision, the account, the device, the counterparty, lets you answer a regulator asking for accountability. The most recent rules on data access go further, they introduce the right to contest an automated decision and to ask for a human review, which makes traceability an obligation before it is an advantage. And the corporate governance codes ask boards to be able to explain how the systems the company relies on work. Quality makes an answer correct, structure makes it defensible, and before a regulator the difference between the two is everything.

13 From the problem to the structure

There remains the most concrete thing of all, the one every real project opens and closes with: where do you start. And the answer I have arrived at [from the tables where I work](#) is almost the opposite of what you would expect from an article that has just spent fifteen thousand words on the elegance of ontologies. You do not start from the ontology. You start from the business problem, the one that keeps management awake at night, and you work back from there to the structure you need to dissolve it. A knowledge graph is justified by the problem it solves, never by the elegance of the graph, and counting the entities inside the graph is not a result, cutting the time it takes to answer a business question is.

The first job, before any graph, is an honest diagnosis of data maturity. Availability, quality, accessibility, governance, and above all fragmentation, how many different sources, how many systems that call the same thing by different names, how much knowledge lives only in people's heads. If data maturity is low and the sources are scattered across seven systems with no consolidation whatsoever, that is the thing to address first, not a showcase predictive use case. A knowledge graph, in this picture, is almost always a foundational enabler, the piece the others lean on, not the first quick win to show a committee to collect

applause.

The numbers on this point are merciless and must be stated plainly, because they dismantle the illusion that the problem is the model. About eighty percent of a data scientist's work goes into data cleaning, entity resolution, schema mapping, not into modeling, the hidden tax you pay when the semantics is implicit and inconsistent. And in a 2025 survey of leaders of enterprise AI programs, two-thirds cite data inconsistency and the lack of semantic alignment as the number-one barrier to reaching production, above any model or infrastructure problem. The classic case is the customer identifier called `customer_id` in one system, `user_id` in another, `id` in a third, and no amount of compute that solves what is, at root, a lack of semantic coherence. Adding bigger models on top of this layer does not cure it, it amplifies it, because an agent takes that inconsistency and propagates it at machine speed.

It helps to distinguish use cases by their nature, because they are not funded and not communicated in the same way. There are the quick wins, high readiness, value in a few weeks, the ones that build trust and momentum. There are the strategic cases, high impact but a horizon of six or twelve months. And there are the foundational cases, the cross-cutting enablers, the prerequisites the others rest on, and the knowledge graph almost always lives here. The typical problem is precisely this, a foundational enabler has the slow pace and deferred value of an infrastructure, while the expectations around AI demand the fast pace and immediate value of a quick win. Whoever funds a knowledge graph expecting applause in the first quarter abandons it in the second, whoever treats it for what it is, a foundation, reaps the harvest on the cases that foundation enables in the following twelve or eighteen months. Knowing which of the three categories you are placing a project in is half its destiny.

From here the path has a shape that real experiences draw fairly consistently. You start from a circumscribed, high-value domain, modeling the essence of the business, a customer identity, a product hierarchy, a fraud-detection case, and eliminating there polysemy, one term with multiple meanings, and synonymy, multiple terms for one meaning. You demonstrate the return on that domain before broadening. Then you break the silos, taking the ontology to the enterprise level and mapping the existing systems onto it, so that queries cut across functions without hand-made integrations every time. Then you iterate on the knowledge, letting algorithms over the graph surface implicit relations and hidden segments. And finally, more a frontier than a stage reached, the graph becomes the reasoning engine the models connect to, and they stop guessing and query a deterministic structure. This arc requires twelve to eighteen months of focused investment to reach maturity, and anyone promising to compress it into a quarter is selling something else.

The maturity journey

From a circumscribed domain to the reasoning engine. Twelve to eighteen months, not a quarter.

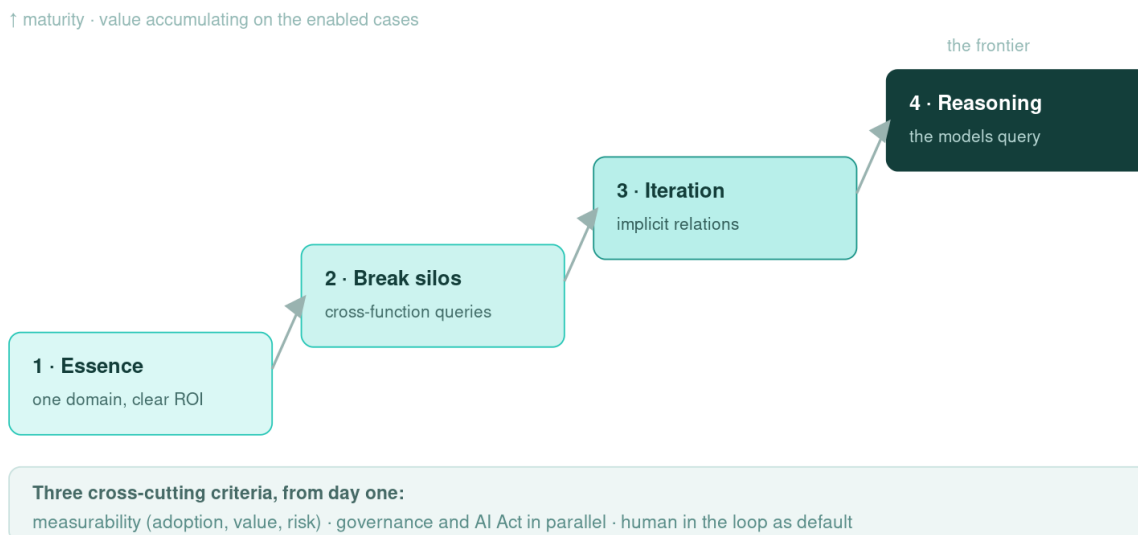


Fig. 12 – The maturity journey.

Every stage has an outcome you can show, and this keeps the program's funding alive. The first produces an ontology-backed application with a clear return on a single domain, and a repeatable model for broadening out. The second produces queries that cut across functions without building a bespoke integration every time. The third surfaces implicit relations and predictions as new leads, things nobody asked for because nobody knew they could be asked. The fourth, the frontier, closes the loop, the AI-validated insight re-enters the ontology as a new rule and the system learns about itself. The returns told by those who have reached the first stages are not symbolic, an industrial operator that grows its core business by more than ten points, an energy network that runs root-cause analysis in seconds instead of days, a service that goes from days of investigation to twenty seconds to trace the root of a problem. You do not need to have the complete ontology tomorrow, you need to start from a circumscribed, high-impact domain, demonstrate the value, broaden from there. Most organizations are at the first two stages, and that is perfectly fine, as long as they know where they are and where they want to get to.

Above this path run three criteria, distinct from the phases because they hold in each, and they must be kept from day one. The first, measurability: no use case starts without a measured baseline and an explicit target, because if you do not know where you were before you cannot claim to have improved, and the metrics must be taken across three families, adoption, value and risk, never adoption alone, because "the AI used five thousand times" is not a result. The second, governance: committee, roles, policies, and for high-risk cases the AI Act compliance check must be set up in parallel with the first use case, not the day after, because governance chased after costs double. The third, the human in the loop as the default: explicit human control at least during rollout, with validation before sending, confidence scores, automatic escalation, especially where there are effects on customers, employees or legal consequences.

A few recurring traps I have seen sink good projects, and recognizing them in time is worth more than any methodology. The roadmap written before the diagnosis, which produces lists of use cases disconnected from the real maturity. The proof of concept that never dies, kept alive for months without an explicit go or no-go, when it should be closed with a binary decision within a few weeks. The vanity metrics, counting

how many times a tool has been used instead of the time saved or the errors avoided, which here means counting the entities in the graph instead of measuring how much faster you answer a question that matters. And the jargon that excludes, running the program in a language for insiders, embeddings, reasoners, retrieval, which cuts out precisely the business leaders who should be steering it. An article on ontologies and knowledge graphs runs the same risk, and that is why I have tried to keep it readable by those who have to steer it, not only by those who implement it.

To understand whether the effort concerns you now or you can postpone it, five signals distilled from enterprise experiences work as a self-diagnosis, the fragmentation of data across multiple systems, high compliance requirements, long decision chains, strong data heterogeneity, the high cost of a wrong answer, and if you recognize three or more, the ontology becomes a competitive necessity more than a laboratory luxury, and the alternative, continuing to build bespoke bridges between one silo and another, one at a time, costly and fragile, is the road that looks cheaper today and turns out to be the most expensive tomorrow. Semantic consistency, at that point, stops being a project and becomes a function of the platform.

Three types of organization adopt the ontology at scale before the others, and it is worth knowing whether you recognize yourself. The platforms with very many users, where every new feature, without a unified ontology, forces you to reintegrate the graph from scratch and fraud rings exploit the cracks between one piece and another. The heavily regulated industries, banks, healthcare, insurance, that must explain their own decisions by tracing the chains of relation among accounts, devices, counterparties. And the complex operational networks, supply chains, logistics, energy grids, telecommunications, where root-cause analysis goes from days to seconds when the value lives in the relations and not in the individual records. The thread that unites them is the same, the value sits in the relations, not in the isolated data, and the more your business resembles a network, the sooner the ontology stops being a luxury.

14 The latent graph

I come back to the word "customer" I opened with, because now that scene reads differently. The three definitions that coexist at the table between sales, finance and support are the ontology of your company as it is right now, implicit, incoherent, never written down, and precisely for that reason impossible to govern. The latent graph you do not have to create from nothing, it already exists, distributed in people's heads and in the joints between systems, and every day it makes decisions in your place.

For years you could afford it, because reading that data there were people who mended the ambiguity with common sense, who knew which "customer" was being talked about and adjusted on the fly. Common sense, inside a model, is not there. An agent inherits your definitions just as they are, coherent or contradictory, written or left to chance, and applies them at the speed and scale of a machine, multiplying the order you put in or the disorder you left.

You might as well choose those definitions now, while the people rereading them are still able to correct them. The moment agents will begin to reason in earnest over your data is not far off, and on that day the latent graph will become your real graph, for how you will have cultivated it or for how you will have let it grow on its own.

Fabio Lalli

Entrepreneur, advisor and speaker on AI and innovation. ZeroFive.AI