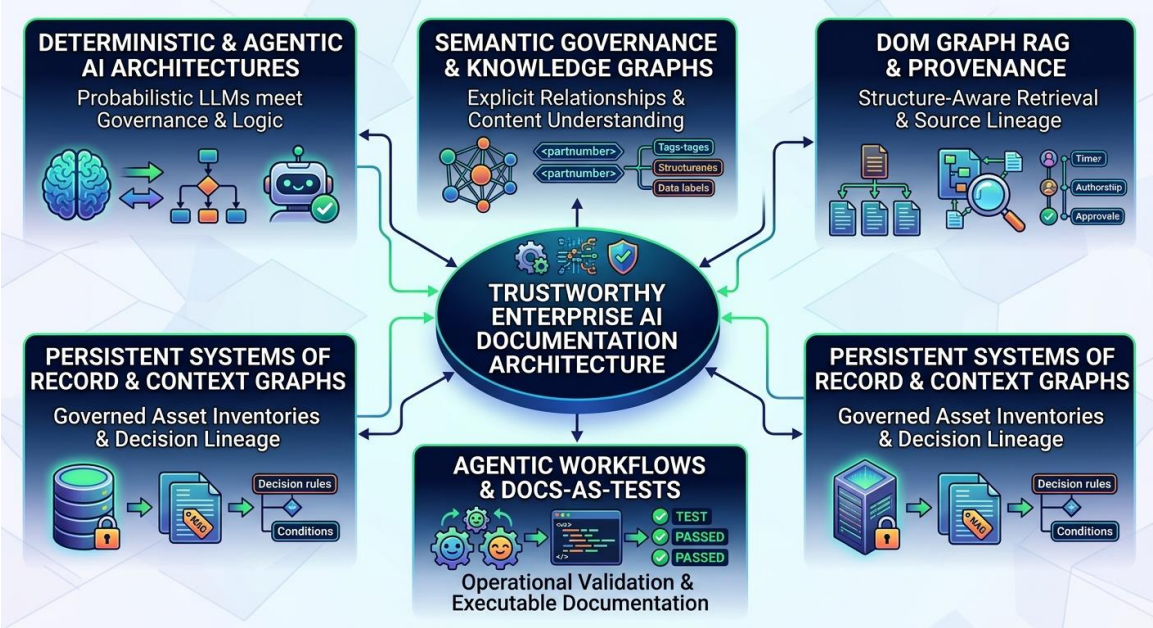
Deterministic and Agentic AI Architectures for Technical Documentation

Semantic Governance, Knowledge Graphs, Context Graphs, DOM Graph RAG, and Executable Validation for Enterprise Documentation Systems

MICHAEL IANTOSCA
SENIOR DIRECTOR OF KNOWLEDGE PLATFORMS AND ENGINEERING
AVALARA



Introduction

Technical documentation organizations are entering one of the most significant transitions in the history of information management. For decades, documentation teams focused primarily on authoring, editing, publishing, and maintaining information intended for human readers. Today, documentation increasingly functions as operational knowledge consumed not only by people, but also by AI systems, support automation, agentic workflows, retrieval engines, semantic search systems, observability platforms, and intelligent customer-assistance applications.

At the center of this transformation is the rise of large language models (LLMs). These systems can summarize engineering specifications, generate release notes, explain APIs, draft troubleshooting procedures, and answer technical questions conversationally with astonishing fluency and confidence. Yet organizations are simultaneously discovering that fluency is not the same thing as truth. LLMs can produce persuasive explanations containing subtle inaccuracies, omitted prerequisites, invented parameters, inconsistent terminology, or operationally dangerous assumptions.

The issue is not that LLMs are unintelligent. The issue is that they are fundamentally *probabilistic* systems. They predict statistically likely language patterns based on training data and inference context. They do not inherently understand whether a rollback procedure is missing, whether a warning applies only to a specific deployment model, or whether a security procedure was deprecated three releases ago.

Human technical writers operate within systems of organizational context, editorial governance, engineering relationships, and operational accountability. Enterprise AI systems require those same structures to be modeled explicitly if they are expected to behave reliably at scale.

This paper explores how deterministic architectures, semantic governance, structured content, knowledge graphs, DOM Graph RAG, iiRDS metadata frameworks, context graphs, agentic Docs-as-Tests validation, provenance tracking, and operational orchestration can mitigate the probabilistic weaknesses of AI systems in enterprise documentation environments.

The Limits of Large Language Models in Enterprise Documentation

Many organizations begin their AI documentation journey by throwing gigantic prompts into an LLM and hoping more context magically produces better answers. At first glance, that sounds reasonable. In practice, it often produces the opposite.

Enterprise documentation ecosystems are extraordinarily large and semantically complex. They may contain millions of words distributed across thousands of guides, APIs, troubleshooting workflows, deployment variants, reusable snippets, metadata structures, release histories, and compliance requirements.

If you doubt this, try a practical experiment. Upload a 300–500 page enterprise document into your favorite LLM and ask it to identify terminology inconsistencies, duplicated guidance, passive voice, or procedural gaps while generating a downloadable report with citations. Then repeat the same task using smaller logical chunks with intermediate summarization or indexing. The difference between advertised context-window size and reliable large-scale reasoning becomes painfully obvious very quickly.

As prompts grow, models struggle with attention allocation, retrieval consistency, instruction persistence, citation fidelity, and cross-document coherence. Important constraints become diluted inside surrounding text. Contradictory product-version guidance may coexist inside the same inference window. Semantically similar but operationally unrelated procedures begin interfering with each other.

The result is what I've jokingly started calling "Splenda reasoning" – output that cosmetically resembles deterministic logic while remaining fundamentally probabilistic underneath. It looks convincing. It sounds authoritative. It may even produce decent prose. But statistical token prediction wearing a lab coat is still statistical token prediction wearing a lab coat.

And, of course, it is expensive. The more work organizations dump into a model, the larger the bill becomes. Vendors are generally delighted to help customers discover this experimentally, if ever.

Technical documentation introduces additional complexity because relationships between topics are rarely linear. A small engineering change may simultaneously affect installation guides, API references, migration procedures, troubleshooting workflows, compliance documentation, cloud deployment guidance, and customer training materials. Different writers may describe identical operational behavior differently while still remaining technically correct within their own contexts.

Organizations therefore need systems capable of reasoning not only about semantic similarity, but also about structure, provenance, metadata, hierarchy, governance rules, operational state, and contextual relationships.

Semantic Markup and Deterministic Content Architecture

One of the strongest deterministic methods available to enterprise documentation organizations is structured semantic content. Structured content organizes information into semantically labeled components rather than loosely formatted narrative text. That distinction matters enormously because enterprise AI systems depend heavily on deterministic preprocessing, metadata governance, provenance tracking, structural reasoning, and retrieval precision long before anything ever reaches a large language model.

A machine cannot reliably infer that the boldfaced string "TD3421" represents a part number. However, when the same value is expressed semantically as <partnumber>TD3421</partnumber>, its meaning becomes explicit, machine-readable, and operationally reusable throughout indexing, retrieval, validation, orchestration, taxonomy management, and governance pipelines.

This is one of the reasons DITA XML remains so valuable in enterprise AI environments. DITA is not merely a publishing format. It is a structured semantic architecture composed of explicitly nested containers with deterministic boundaries, inheritance behavior, metadata scope, and operational semantics.

That containment model turns out to be incredibly important for AI systems.

A DITA task topic does not simply “look like” a procedure to a human reader. The architecture explicitly defines the content as a procedure containing prerequisites, ordered steps, warnings, rollback conditions, expected results, references, reusable content, metadata relationships, and contextual boundaries. AI pipelines therefore do not need to guess which content belongs together operationally because the source structure already encodes those relationships directly.

This becomes especially important during preprocessing and retrieval orchestration. Modern RAG pipelines frequently normalize, transform, flatten, enrich, or selectively strip portions of markup before inference occurs. The primary advantage of semantic markup is therefore not that the LLM directly consumes raw XML. In many production systems, the model may receive mostly normalized text.

The real value lies upstream.

Structured semantic architectures allow preprocessing systems to perform deterministic chunking, metadata enrichment, provenance-aware retrieval, relationship traversal, taxonomy propagation, update targeting, inheritance resolution, and context assembly with far greater precision than loosely structured text environments. The semantic intelligence encoded in the source architecture tells the system which content elements must remain together, which metadata applies contextually, which relationships must be preserved, and which operational boundaries should remain isolated.

In effect, the semantic structure acts as a governance and orchestration layer for the retrieval system itself.

This is where DITA differs substantially from presentation-oriented formats such as Markdown, AsciiDoc, or ReStructuredText. Those formats are excellent for lightweight publishing and developer-centric workflows, but semantically they are still primarily formatting-oriented text unless heavily extended with additional structural and metadata layers.

DITA content, by contrast, is inherently containerized. Metadata and taxonomy labels can cascade predictably through nested child structures without contaminating unrelated peer content. A deployment-environment taxonomy attached at a parent procedural level, for

example, can inherit downward into subordinate elements while remaining isolated from adjacent procedures or unrelated topic branches.

Achieving equivalent semantic containment and inheritance behavior in lightweight markup ecosystems often requires substantial custom engineering – heuristic parsing, regex segmentation, AST reconstruction, metadata overlays, custom governance pipelines, or external semantic augmentation layers. At enterprise scale, those approximations become increasingly brittle and operationally expensive.

For years, many people argued that DITA XML was getting "long in the tooth" compared to lightweight authoring formats because simpler human editing was assumed to be the future optimization target. AI changed that equation rather dramatically.

AI systems do not care whether a format feels elegant in a text editor. They care whether the surrounding architecture can deterministically preserve semantic meaning, structural boundaries, metadata inheritance, provenance, operational context, and governance constraints across retrieval, orchestration, validation, and automated update workflows.

As organizations move toward agentic AI systems, that distinction stops being philosophical and starts becoming architectural.

Knowledge Graphs, DOM Graph RAG, and Provenance

Knowledge graphs have emerged as one of the most important deterministic technologies in enterprise AI architectures because they model semantic relationships explicitly rather than inferring them statistically.

A knowledge graph may represent that a troubleshooting topic applies only to a specific deployment model, that a procedure references a particular API, or that a feature was introduced in a certain release. This explicit relationship modeling enables deterministically governed reasoning and impact analysis across enormous documentation ecosystems.

Equally important is provenance. Organizations need to know not only what information an AI system retrieved, but where it came from, who approved it, whether it is current, and whether it is operationally authoritative. Without provenance-aware retrieval, AI systems frequently combine current engineering guidance with deprecated release notes, emergency workarounds, or obsolete customer instructions.

Traditional vector-based RAG improves semantic retrieval, but vector embedding frequently strips away structural information. A warning paragraph may remain

semantically retrievable while losing the procedure hierarchy, conditional applicability, metadata lineage, or product-version context surrounding it.

Document Object Model RAG – usually shortened to DOM Graph RAG – addresses this by preserving document structure during retrieval. Rather than treating documentation as disconnected chunks, DOM-aware retrieval understands hierarchical relationships between sections, procedures, warnings, tables, reusable snippets, metadata inheritance, XML structures, and conditional content.

When combined with iiRDS metadata frameworks and knowledge graphs enriched with domain-specific semantics, retrieval systems gain the ability to traverse semantic meaning and document structure simultaneously.

Suppose a Kubernetes authentication enhancement affects only cloud deployments using a specific identity provider. A DOM Graph RAG system can isolate the exact subsections requiring updates while preserving unaffected procedural content and maintaining source lineage. That level of precision is extraordinarily difficult using manual search, keyword matching, or vector similarity alone.

Context Graphs and Agentic Governance

Knowledge graphs model semantic truth effectively, but agentic systems also require operational governance structures capable of representing state, legality conditions, approvals, escalation rules, confidence thresholds, sequencing dependencies, and execution context. This is where context graphs become critically important.

A context graph extends semantic modeling into operational governance. While a knowledge graph may represent that a procedure references a particular API, a context graph models the conditions under which an agent is allowed to act, publish, escalate, retry, block execution, or request human review.

Think of it this way – a context graph functions less like a static workflow diagram and more like a constitution for agent decision-making and orchestration.

It does not merely describe what agents can do. It models when, why, and under what conditions actions are permitted, prohibited, escalated, retried, or invalidated. Relationships inside the graph may represent concepts such as approval requirements, rollback conditions, compliance restrictions, publication eligibility, confidence thresholds, or policy violations.

This matters because enterprise documentation governance is rarely linear. A workflow triggered by an engineering change request may involve ingestion, semantic validation, document-model conformance checking, AI draft generation, Docs-as-Tests execution,

compliance review, human approval, and publishing orchestration. Different product areas may require entirely different governance conditions.

Without explicit contextual governance, these conditions become buried inside prompts, orchestration code, or tribal knowledge. Systems become brittle, difficult to audit, and nearly impossible to explain after failures occur.

Context graphs externalize governance relationships into explicit semantic structures. Agents can reason dynamically about operational legality rather than blindly following procedural scripts.

Critically, context graphs can also capture runtime execution telemetry and decision lineage. As agents operate, the system can record which constraints were evaluated, which decisions were made, which policies were triggered, what confidence signals existed, and why particular actions occurred. This creates a closed, auditable governance loop that supports traceability, explainability, compliance validation, and continuous refinement over time.

In effect, the context graph becomes both the governance model and the operational memory of the agentic system.

Agentic Workflows and Docs-as-Tests

Enterprise documentation maintenance is fundamentally a multi-stage operational process rather than a single generative event. Human technical writers naturally ingest source material, identify missing information, infer dependencies, validate terminology, review procedures, test workflows, and refine outputs iteratively.

Agentic architectures externalize many of these responsibilities into specialized cooperating agents. Mature workflows may include ingestion agents, metadata-analysis agents, graph-reasoning agents, topic-impact-analysis agents, draft-generation agents, semantic-validation agents, Docs-as-Tests agents, analytics agents, and publishing-governance agents.

The ingestion stage is particularly important because AI systems frequently attempt to generate documentation from incomplete engineering input. Human writers instinctively recognize missing prerequisites, rollback steps, unsupported platform details, environmental constraints, or dependency conditions. AI systems require explicit mechanisms for identifying those same gaps. Some of that missing governance can be partially automated and mitigated with input knowledge validation agents that perform assessment against detailed criteria and scoring.

Even with input validation agents, Docs-as-Tests represents one of the strongest deterministic safeguards within these ecosystems. Rather than assuming documentation is correct because humans reviewed it, organizations can design and include agents that execute the documentation itself for operational validation.

Deployment guides run against temporary cloud environments. API examples execute against test systems. Configuration procedures validate against live infrastructure. Troubleshooting workflows execute against known failure conditions.

This transforms documentation from static prose into executable operational knowledge that can also become an operational system-of-record. Now that the input to agentic SDLC is essentially rich documents, validated documentation can be used as part of the input corpus to agentic SDLC workflows – or what we might call "closed-loop operational documentation ecosystems" or "Docs as Code." Ponder that notion for a few moments.

A Kubernetes deployment guide provides a useful example. An agentic Docs-as-Tests workflow may provision temporary infrastructure automatically using Terraform and Kubernetes orchestration, execute documented procedures step-by-step, validate commands, detect missing prerequisites, capture failures, and feed operational analytics back into governance systems.

The result is not merely polished prose. It is documentation that has empirically demonstrated operational validity.

Technologies Supporting Deterministic Documentation Ecosystems

A growing ecosystem of technologies now supports deterministic and agentic documentation architectures capable of semantic reasoning, operational validation, explainable governance, and autonomous orchestration.

Workflow orchestration platforms such as Temporal coordinate long-running stateful workflows involving retries, approvals, asynchronous processing, validation loops, and rollback handling. LangGraph provides graph-based orchestration capabilities allowing specialized agents to coordinate dynamically while preserving execution lineage and decision context. Other frameworks such as Semantic Kernel, CrewAI, AutoGen, Haystack, and LlamaIndex are increasingly being explored for governed multi-agent systems.

At the semantic layer, organizations are adopting ontology and knowledge-graph platforms such as PoolParty and GraphDB to build governed semantic ecosystems based on RDF, OWL, SPARQL, RDFS, and SKOS standards.

These systems are important because they enable deterministic inference. Unlike probabilistic vector retrieval systems that infer relationships statistically, OWL-based reasoning can derive new facts deterministically from explicitly modeled rules and relationships.

Inference engines can automatically identify impacted procedures, detect dependency conflicts, validate metadata relationships, infer operational constraints, and traverse semantic hierarchies in ways that remain explainable and auditable.

Pre- and post-generation linting systems such as Acrolinx and its successor Markup AI add additional governance layers for terminology control, readability validation, semantic consistency, and AI-oriented structural validation.

Docs-as-Tests automation frameworks such as Playwright, Cypress, Selenium, Postman, pytest, and Robot Framework strengthen deterministic validation by operationally executing documented procedures rather than reviewing them purely as narrative instructions.

The most advanced enterprise architectures increasingly combine all of these technologies into unified semantic-governance ecosystems. Knowledge graphs identify affected documentation relationships. DOM Graph RAG preserves structural lineage. Context graphs govern operational legality and escalation behavior. Validation agents enforce DITA structural compliance. Docs-as-Tests agents verify operational correctness empirically. Observability systems capture runtime telemetry, validation outcomes, governance metrics, and agent decision lineage.

The result is not simply faster content generation. The result is measurable documentation reliability, semantic traceability, deterministically governed reasoning, explainable governance, and operational trustworthiness.

Agentic Content Governance and Persistent Systems of Record

Enterprise-scale AI documentation systems require more than repositories of content and agentic content creation. They require centralized operational governance and persistent systems of record (SoR) capable of maintaining authoritative inventories of documentation assets, semantic relationships, governance states, lifecycle metadata, workflow dependencies, and organizational ownership structures over time.

Most documentation environments today remain fragmented across CCMS platforms, repositories, support portals, LMS systems, localization pipelines, PDF archives, and marketing ecosystems. Agentic systems require something more unified.

A mature architecture therefore includes governed document inventories in which every asset maintains rich operational metadata including ownership, publication status, approval history, validation state, dependency relationships, translation requirements, regulatory classifications, lifecycle alignment, and observability metrics.

This persistent operational context becomes critically important because autonomous systems must understand not only what content exists, but whether it is active, deprecated, regulated, validated, approved, or operationally safe to modify, sometimes over many years.

Persistent inventories also allow organizations to model documentation as part of broader customer information ecosystems rather than isolated deliverables. Once semantic relationships are tracked centrally, organizations can identify gaps, redundancies, inconsistencies, outdated workflows, and friction points across entire customer journeys.

Over time, these governance architectures evolve into something fundamentally different from traditional publishing systems. Traditional systems govern files and publishing workflows. Mature *agentic governance systems* govern operational knowledge ecosystems.

Putting It All Together

The future of enterprise technical documentation will not belong to organizations that merely generate more content with AI. It will belong to organizations that build semantically governed, operationally validated, and explainable knowledge ecosystems around AI generation.

Large language models are remarkable language-generation systems, but they remain fundamentally probabilistic, and no amount of vector-based probabilistic augmentation, recursive prompt gymnastics, or trillions of additional parameters magically transforms probabilistic token prediction into deterministic operational intelligence – regardless of what the AI snake-oil salesmen on LinkedIn insist between inspirational rocket-ship emojis. LLMs predict statistically likely outputs. They do not inherently understand operational correctness, governance policy, procedural safety, rollback integrity, regulatory compliance, or whether the “helpful” configuration change they just suggested is going to quietly detonate a production Kubernetes cluster at 2:13 a.m. while everyone is asleep and the on-call engineer is reconsidering their career choices.

That is not a moral failure of AI. It is simply the architectural reality of probabilistic systems pretending to perform deterministic operational reasoning often enough to make people dangerously optimistic.

This is precisely why deterministic models and governance matter.

Structured content, semantic markup, metadata governance, provenance tracking, DOM Graph RAG, iiRDS frameworks, knowledge graphs, RDF and OWL ontologies, context graphs, deterministic inference engines, orchestration platforms, Docs-as-Tests automation, and runtime observability together create something fundamentally different from prompt engineering. They create governed operational ecosystems capable of supporting trustworthy enterprise AI at scale.

Most importantly, these architectures externalize the kinds of reasoning experienced technical writers, information architects, and operational engineers already perform instinctively. Human experts naturally reason about dependencies, procedural risk, audience context, lifecycle state, semantic relationships, compliance obligations, and operational safety. Mature agentic architectures do not replace those governance models. They formalize, operationalize, and scale them.

A lot of organizations are currently approaching enterprise AI documentation as though success simply requires larger context windows, larger prompts, and increasingly heroic cloud-inference invoices. Unfortunately, dumping an entire documentation repository into an LLM and hoping statistical token prediction somehow evolves into governance is less an architecture strategy and more a budgetary cry for help.

The organizations that succeed in this transition will recognize that trustworthy AI documentation is not fundamentally a content-generation problem. It is a semantic-governance, operational-validation, deterministic-reasoning, and explainability problem.

In the end, the winners probably will not be the organizations with the prettiest AI-generated prose. They will be the organizations whose systems can actually explain why a decision was made, prove that it was operationally valid, trace where the information originated, enforce governance constraints consistently, and avoid rewriting half the documentation repository because a probabilistic model became creatively confident on a Tuesday afternoon.