LAYERONE

# GUIDE: AGENTIC COMMERCE PATTERNS

A PRACTICAL GUIDE TO HOW AI ACTUALLY GETS WORK DONE IN B2B COMMERCE

March 2, 2026

# CONTENTS

# INTRODUCTION: FROM APIS TO AGENTS

For the last two decades, digital commerce has been built on a predictable foundation: structured APIs, deterministic workflows, and tightly controlled user interfaces. Every interaction was designed, every path anticipated.

**Agentic commerce breaks that model.**

APIs do not disappear. They become the foundation. But instead of being directly orchestrated by rigid workflows, they are invoked dynamically by intelligent systems that can reason about intent.  This introduces a new architectural layer.

Between the user and the APIs sits an agent that can:

- Interpret intent
- Plan a sequence of actions
- Invoke the right systems
- Validate results
- Adjust based on feedback

This layer is what transforms a collection of services into an intelligent system.

But this shift introduces a new challenge. If agents are doing the work, how exactly are they doing it?

The answer lies in a set of emerging design patterns. These patterns define how agents reason, act, recover, and collaborate. They are the building blocks of agentic commerce systems.

This paper outlines the most important of these patterns and how they apply specifically to B2B commerce environments.

## ABOUT LAYER ONE

Layer One is a consultative solutions partner for distributors, manufacturers, and related verticals, operating wholly in the US. Over the last two decades, we’ve evolved from a traditional agency to experts in the digital ecosystem, solving the unique obstacles companies face with their eCommerce and PIM optimization. We truly partner with our clients, leveraging our innovative approach and versatile team of thought leaders to drive results.

Layer One pushes our clients to think outside of the box. We are always challenging ourselves to be just a little better every day. We bring expertise in marketing, technology, digital strategy, and commerce. We have the skills and experience to help you understand what makes your buyers tick.

Our 100% US team assimilates with yours as a strategic partner, fostering a relationship of trust and collaboration. Our monthly reviews with high NPS ratings, weekly touchpoints, and daily availability means our clients are heard, understood, and confident with the strategic business decisions and processes we build together.

Our focus is on your short and long-term success. So rather than projects that have set finish lines, we integrate with your team in ongoing engagements that evolve as new needs arise and the demands change.

We attack the engagement with an iterative process that demands outcomes from well-defined success criteria. Simply, we start building and creating with you from “GO.”

### *PLATFORM PARTNERS*

With a select set of technology partners that are leaders in their own right, Layer One creates best-in-class, platform-agnostic solutions for our clients. From Optimizely and Sitecore to Shopify and inriver, Layer One has chosen to work with the best of the best.

# THE CORE MENTAL MODEL: THINK IN CAPABILITIES, NOT ENDPOINTS

Before diving into patterns, it is important to shift how we think about system design.

Traditional commerce systems expose endpoints:

- Get product
- Check inventory
- Create order

Agentic systems expose capabilities:

- "Find a suitable replacement for this discontinued part"
- "Build a compliant quote based on customer constraints"
- "Optimize this order for margin and availability"

This shift is what enables agents to orchestrate behavior rather than execute scripts. Instead of following a fixed sequence of steps, the system can determine how to achieve an outcome based on context.

## MENTAL EXERCISE

Remove your entire UI.
No search. No product pages. No cart.

A buyer says:
"Find a compliant replacement, in stock, within budget."

What do you call first?
Which system decides?

**APIs expose actions.**
**Agents fulfill intent.**

This highlights the difference between deterministic and stochastic models. Traditional commerce systems are deterministic, producing the same result for the same input. They are reliable but rigid. Agentic systems introduce stochastic behavior through language models, allowing them to interpret intent and adapt, but with less predictability.

The patterns that follow bridge these two worlds. They bring structure to stochastic reasoning so agents can operate flexibly while still producing consistent, controlled outcomes aligned with deterministic business systems.

# AGENTIC COMMERCE PATTERNS

Most teams approach agentic commerce as a model problem. It is not. It is a systems problem. Without structure, agents are inconsistent, opaque, and unreliable. The difference between a demo and a production system is not intelligence. It is design. The patterns that follow are what turn raw model capability into something that can actually run a business.

## PROMPT CHAINING (DECOMPOSITION OF WORK)

The prompt chaining pattern enables an agent to handle complex tasks by breaking them into a sequence of smaller, manageable steps. Rather than attempting to solve a multi-faceted problem in a single pass, the agent decomposes the work into stages, where each step produces an output that informs the next. This allows the system to progressively refine understanding, reduce ambiguity, and maintain control over intermediate results. In practice, prompt chaining introduces structure to otherwise open-ended reasoning. It makes the agent's behavior more transparent, easier to debug, and more reliable in production environments. By treating each step as a discrete unit of work, the system can validate, adjust, or retry along the way, preventing errors from cascading through the entire process.

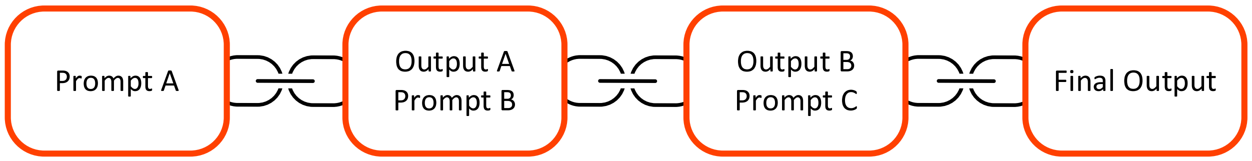

*Figure 1 - Prompt Chaining*

### *WHY IT MATTERS IN COMMERCE*

B2B commerce tasks are rarely atomic. Consider:

"Find me an equivalent cable to this SKU, ensure it meets spec, check availability across warehouses, and generate a quote."

This cannot be solved in a single step reliably. It requires decomposition.

## HOW IT WORKS

A typical chain might look like:

1. Interpret intent
2. Extract product requirements
3. Search for candidate SKUs
4. Validate specifications
5. Check availability
6. Generate pricing
7. Assemble response

Each step becomes its own prompt or tool call.

## WHERE IT BREAKS

- Error propagation across steps
- Loss of context between steps
- Latency from too many chained calls

## COMMERCE INSIGHT

Prompt chaining maps closely to traditional middleware orchestration, but with probabilistic decision-making at each step. This is where many early implementations stall. Without guardrails, chains become fragile.

## TOOL USE (BRIDGING TO REAL SYSTEMS)

The tool use pattern is what allows an agent to move beyond reasoning and into execution. Instead of relying on its own internal knowledge, the agent invokes external systems to retrieve data or perform actions in the real world. These tools represent structured, deterministic interfaces to systems such as PIM, ERP, inventory services, pricing engines, and ecommerce platforms. The agent determines when a tool is needed, supplies the appropriate inputs, and then incorporates the structured output back into its reasoning process. This creates a clear separation between probabilistic decision-making and reliable system execution. In practice, tool use is what grounds agent behavior in reality. It ensures that availability is current, pricing is accurate, and transactions are valid. Without it, agents speculate. With it, they operate.

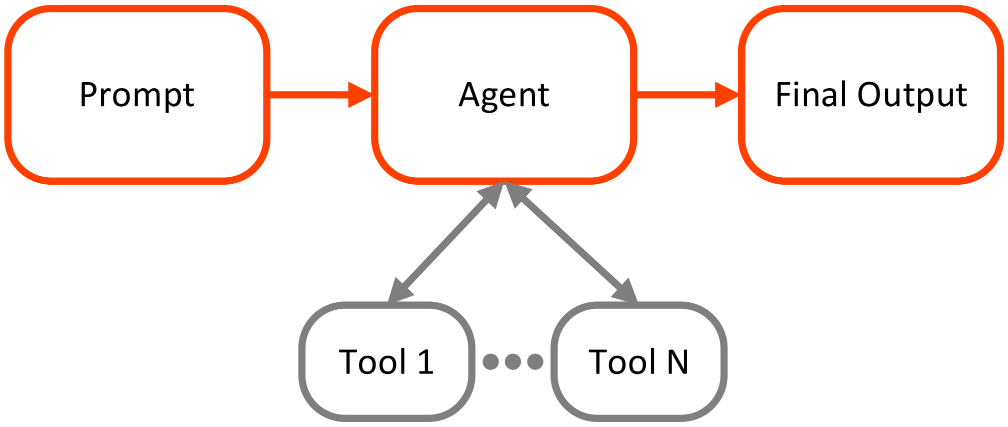

*Figure 2 - Tool Use*

### WHY IT MATTERS IN COMMERCE

Without tools, agents hallucinate. With tools, they transact.

<u>Examples:</u>

- Checking real-time inventory before confirming availability
- Creating a cart or checkout session
- Retrieving customer-specific pricing

### DESIGN CONSIDERATIONS

- Tools must be deterministic
- Inputs must be strongly typed
- Outputs must be structured and validated

### COMMERCE INSIGHT

Tool design is the new API design. If your tools are not clean, your agent will not be reliable. This is where protocols like ACP begin to matter, as they standardize how agents interact with commerce systems.

## SKILL USE (REUSABLE CAPABILITIES)

The skill use pattern introduces a layer of abstraction that allows agents to operate in terms of reusable business capabilities rather than individual prompts or tool calls. A skill encapsulates a defined outcome, such as product substitution, attribute normalization, or quote generation, and bundles together the underlying logic, tools, and decision rules required to achieve it. Instead of orchestrating low-level actions each time, the agent invokes a skill as a cohesive unit, reducing complexity and improving consistency. This pattern is especially important in B2B commerce, where domain knowledge and business rules are critical. By embedding that expertise into skills, organizations can ensure that agents behave in ways that align with real-world processes, while also making those capabilities easier to reuse, govern, and evolve over time.

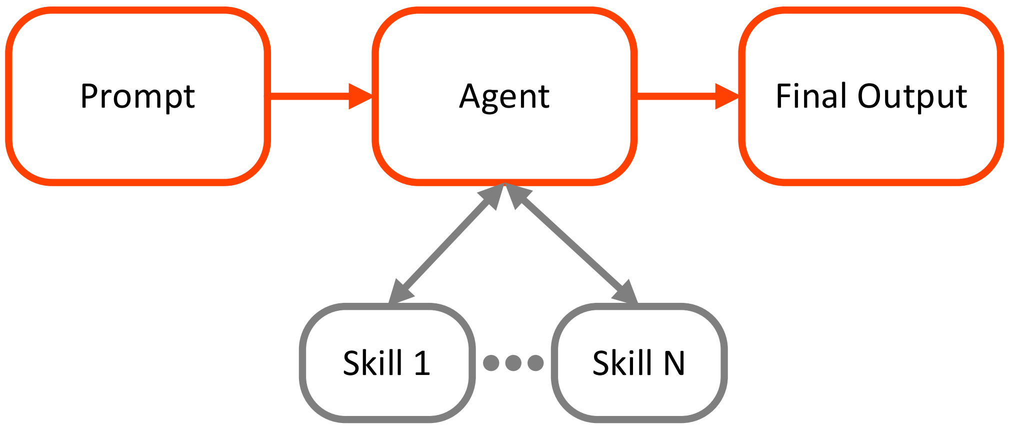

*Figure 3 - Skill Use*

### WHY IT MATTERS

The skill use pattern is where domain knowledge becomes operational. Without it, agents assemble low-level actions each time, which leads to inconsistency. Skills package business logic, rules, and system interactions into reusable capabilities that produce reliable outcomes. This makes behavior more predictable, easier to govern, and aligned with how the business operates, while providing a clear path to scale and evolve processes over time.

### COMMERCE INSIGHT

Skills are where domain expertise lives. In B2B, this is critical. A generic agent cannot understand:

- Spec sheets
- Substitution rules
- Compliance constraints

Skills encode that knowledge.

## REFLECTION (SELF-CORRECTION AND VALIDATION)

The reflection pattern allows an agent to evaluate its own output before taking action or returning a result. After generating a response, the agent reviews it against defined goals, constraints, or quality criteria to determine whether it is complete and correct. If issues are identified, the agent can revise its output and re-evaluate until it meets the required standard. This introduces a feedback loop that improves reliability and reduces the likelihood of errors. In commerce scenarios, where mistakes can have financial or operational impact, reflection helps ensure that outputs such as pricing, configurations, or recommendations are accurate and aligned with business rules.

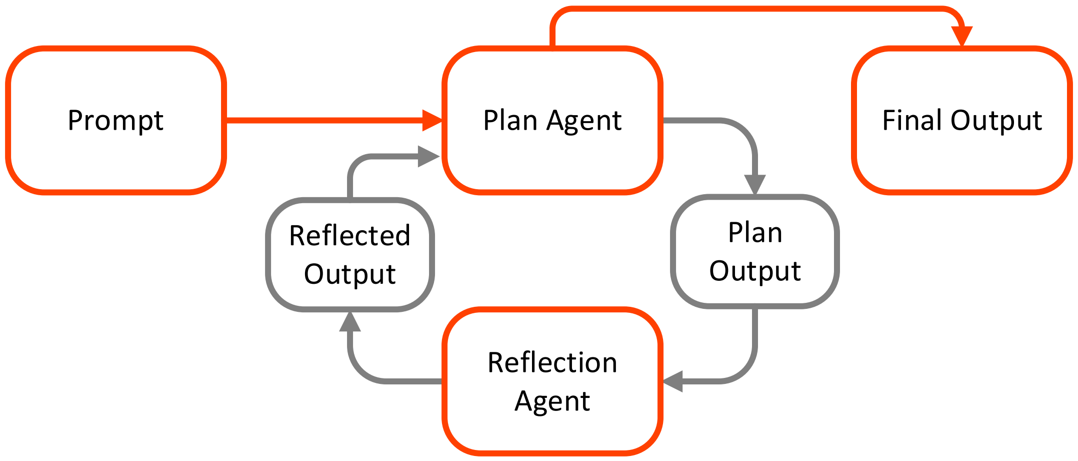

*Figure 4 - Reflection*

### *WHY IT MATTERS*

The reflection pattern improves reliability by adding a feedback loop to agent behavior. Instead of returning a single result, the agent evaluates its output against goals and constraints and revises as needed. This leads to more consistent outcomes and reduces errors before actions are taken.

### *HOW IT WORKS*

After generating a response, the agent:

1. Reviews the output
2. Checks against constraints
3. Revises if needed

### *COMMERCE INSIGHT*

Reflection is essential for trust. In many cases, it replaces traditional validation layers. However, it must be paired with deterministic checks for critical operations.

## PLANNING (GOAL-ORIENTED EXECUTION)

The planning pattern enables an agent to define a structured approach to achieving a goal before executing any actions. Rather than reacting step by step, the agent first outlines the sequence of tasks required, identifies dependencies, and determines the order of operations. This creates a clear path from intent to outcome and reduces the risk of inefficient or incorrect execution. By separating planning from execution, the system gains greater transparency and control, making it easier to adjust strategies when conditions change. In complex commerce scenarios, planning allows agents to handle multi-step requests more effectively by ensuring that each action contributes to a coherent overall objective.

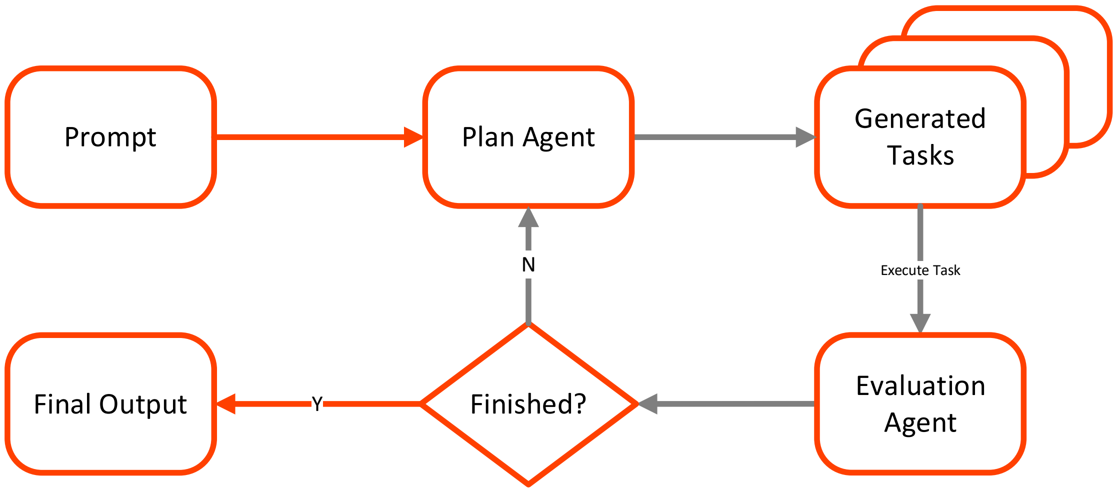

*Figure 5 - Planning*

### *WHY IT MATTERS*

Without planning, agents operate reactively. With planning, they operate strategically.
Example:
User request:“Build me a full order for a new warehouse setup.”
Plan:

1. Identify required product categories
2. Generate bill of materials
3. Validate compatibility
4. Check inventory
5. Optimize for cost and lead time
6. Generate quote

### *COMMERCE INSIGHT*

Planning is what enables agents to handle large, ambiguous B2B requests. It mirrors how experienced sales reps think.

## MULTI-AGENT SYSTEMS (SPECIALIZATION AND COLLABORATION)

The multi-agent systems pattern distributes responsibility across multiple specialized agents that collaborate to achieve a shared goal. Instead of relying on a single general-purpose agent, each agent is designed to handle a specific domain such as product data, pricing, inventory, or compliance. These agents operate with a shared context and exchange information as needed, allowing each to contribute its expertise to the overall outcome. This approach improves scalability and accuracy by aligning tasks with domain knowledge, while also making the system more modular and easier to evolve. In complex commerce environments, where no single model can reliably manage all concerns, multi-agent systems provide a practical way to coordinate specialized decision-making into a cohesive result.

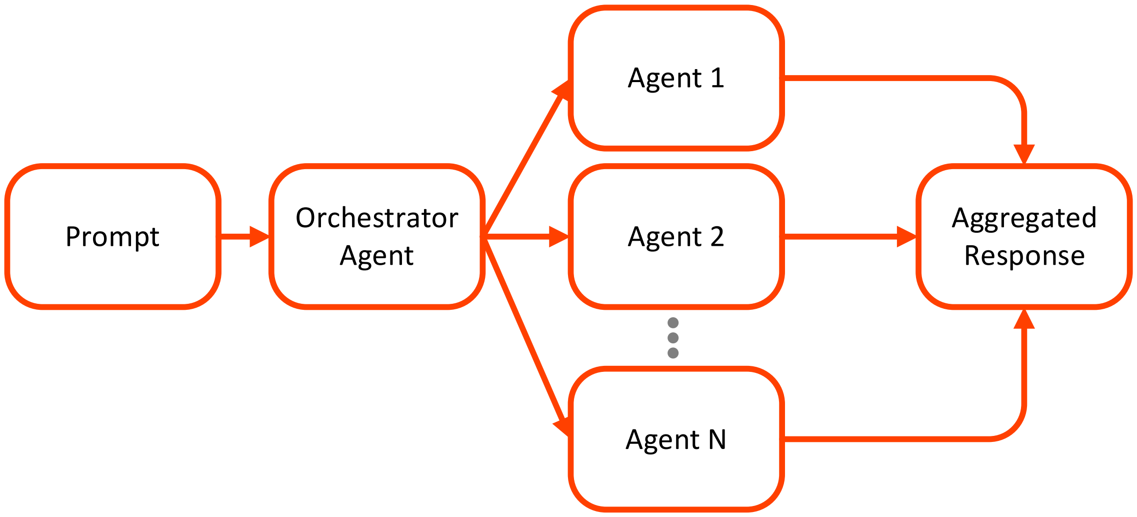

*Figure 6 - Multi-Agent Systems*

### EXAMPLE ROLES

- Product specialist agent
- Pricing agent
- Inventory agent
- Compliance agent

### WHY IT MATTERS

No single agent can effectively handle all domains in complex commerce environments.

### COMMERCE INSIGHT

This pattern mirrors organizational structure. The risk is coordination overhead and conflicting outputs. Strong orchestration is required.

## MEMORY (CONTEXT PERSISTENCE)

The memory pattern allows an agent to retain and use context across interactions, enabling more informed and personalized behavior over time. This includes short-term memory that captures the current session as well as long-term memory that stores historical data such as customer preferences, past orders, and prior decisions. By referencing this context, the agent can avoid redundant work, maintain continuity, and tailor its responses to the specific user or situation. In commerce environments, memory supports more efficient repeat transactions, better recommendations, and stronger alignment with customer relationships. At the same time, it requires careful governance to ensure that stored information remains accurate, relevant, and appropriate for use.

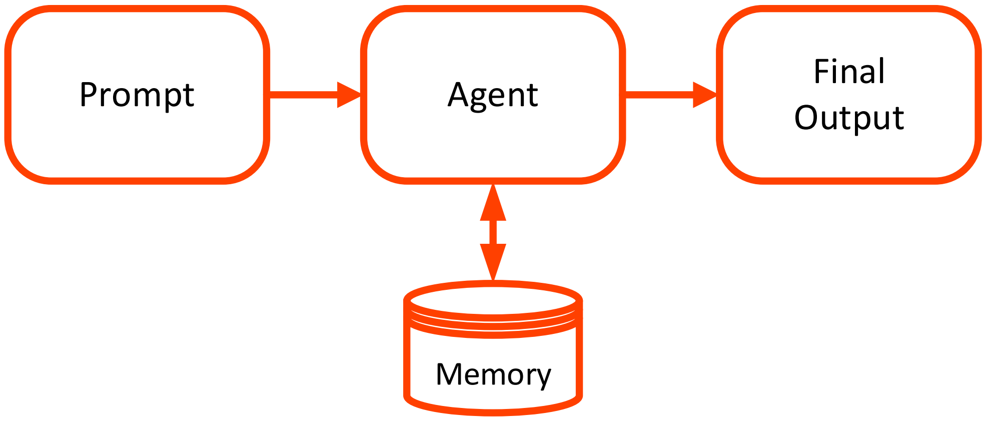

*Figure 7 - Memory*

### TYPES

- Short-term (session context)
- Long-term (customer preferences, past orders)

### WHY IT MATTERS

B2B relationships are not transactional. They are ongoing.

### COMMERCE INSIGHT

Memory enables:

- Personalized pricing
- Repeat order acceleration
- Context-aware recommendations

However, it introduces governance challenges around data accuracy and privacy.

## RETRIEVAL-AUGMENTED GENERATION (RAG)

The retrieval-augmented generation pattern combines language models with external data sources to produce responses grounded in accurate, up-to-date information. Instead of relying solely on what the model has learned, the agent retrieves relevant content from systems such as product catalogs, technical documentation, or knowledge bases and incorporates that context into its reasoning. This approach improves accuracy, reduces hallucination, and ensures that outputs reflect the current state of the business. In B2B commerce, where product data is complex and constantly changing, retrieval-augmented generation is essential for delivering reliable answers and decisions based on real data rather than assumptions.

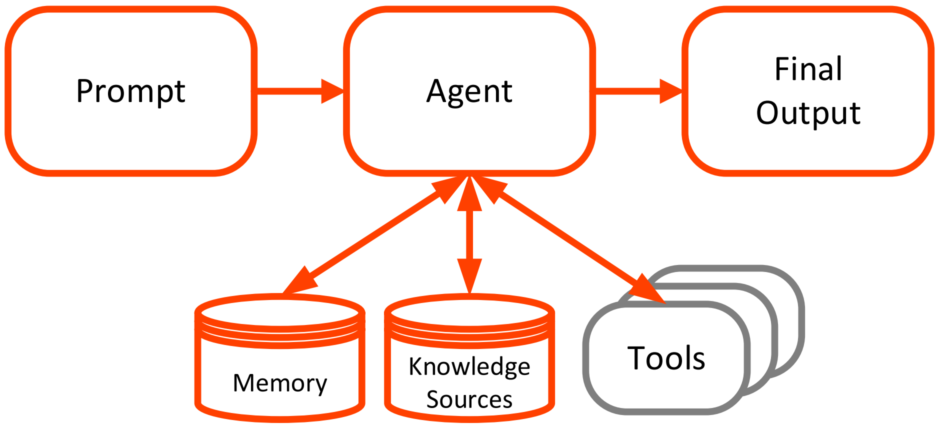

*Figure 8 - RAG*

While this pattern may seem identical to the Tool Use pattern, it solves a different problem.

The tool use pattern is about *taking action and accessing live systems*. When an agent uses a tool, it is calling a deterministic interface to do something real, such as checking inventory, retrieving customer-specific pricing, or creating an order. Tools are transactional and authoritative. They return structured data or perform operations that directly impact the business.

The retrieval-augmented generation pattern is *about informing reasoning with relevant context*. Instead of calling transactional systems, the agent retrieves unstructured or semi-structured information such as product descriptions, spec sheets, or documentation. This information is then used to improve understanding and generate better responses. RAG does not execute actions. It provides context.

A simple way to think about the difference:

- RAG helps the agent know
- Tool use helps the agent do

In practice, both patterns work together. An agent might use RAG to understand what qualifies as an equivalent product, then use tools to validate availability, pricing, and ultimately complete the transaction.

## WHY IT MATTERS

Retrieval-augmented generation connects agents directly to authoritative data sources at the moment a decision is made. Instead of relying on static knowledge, the agent pulls in relevant information from systems like PIM, documentation repositories, and product records to inform its response. This keeps outputs aligned with current specifications, availability, and business context. In B2B environments, where accuracy and detail are critical, this approach reduces risk and ensures that recommendations and decisions are based on verified information rather than inference.

## COMMERCE INSIGHT

RAG is foundational in B2B because product complexity is high. However, raw retrieval is not enough. Data must be normalized and structured, which is where PIM systems become critical.

## CONSTRAINT ENFORCEMENT (GUARDRAILS FOR REALITY)

The constraint enforcement pattern ensures that an agent's decisions remain within defined business rules and operational limits. After generating a proposed action or output, the system evaluates it against constraints such as pricing thresholds, inventory availability, customer entitlements, or compliance requirements. If the result violates any constraint, it is adjusted, rejected, or rerouted for further handling. This pattern provides a critical layer of control that balances the flexibility of agent reasoning with the rigidity required in commerce systems. By enforcing constraints consistently, organizations can trust that agent-driven actions align with real-world rules and do not introduce risk.

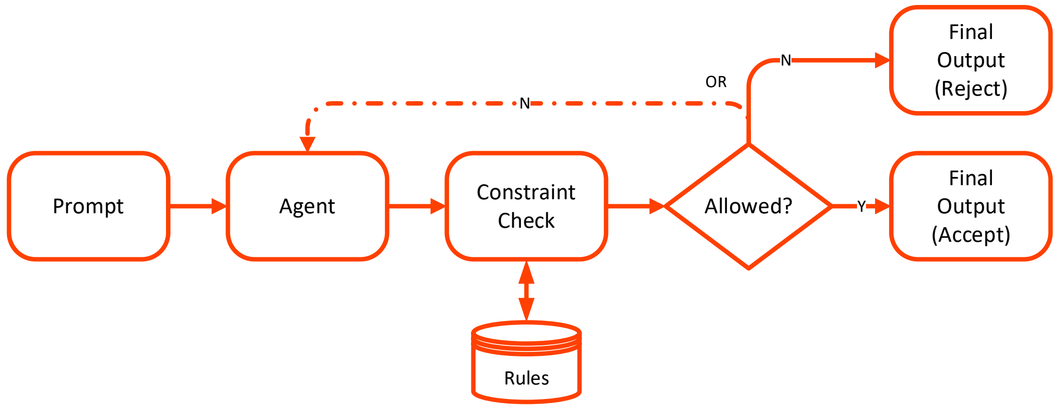

*Figure 9 - Constraint Enforcement*

Once again this may seem identical to the reflection pattern as the constraint enforcement and reflection patterns both evaluate outputs, but they serve different roles.

The reflection pattern is about *self-assessment*. The agent reviews its own output to determine if it is complete, coherent, and aligned with the original intent. It is a qualitative check driven by the model's reasoning, and it may result in revisions to improve the response.

The constraint enforcement pattern is about *rule enforcement*. The system checks the output against explicit, predefined business rules such as pricing limits, inventory availability, or compliance requirements. These checks are deterministic and non-negotiable. In simple terms:

- Reflection asks, "Does this make sense?"
- Constraint enforcement asks, "Is this allowed?"

Reflection improves quality. Constraint enforcement guarantees correctness.

## *WHY IT MATTERS*

Agents are probabilistic. Commerce systems are not.  Constraint enforcement anchors agent behavior to hard business rules and operational limits. Rather than relying on judgment alone, every proposed action is checked against defined policies such as pricing thresholds, inventory availability, customer entitlements, or compliance requirements. This ensures that outcomes stay within acceptable boundaries, regardless of how the agent arrived at them. In B2B commerce, where violations can create financial or regulatory risk, this pattern provides a necessary layer of control that keeps automation aligned with real-world constraints.

## *COMMERCE INSIGHT*

Constraint enforcement is the bridge between AI flexibility and business reality. This is where most production systems either succeed or fail.

## HUMAN-IN-THE-LOOP (ESCALATION AND OVERSIGHT)

The human-in-the-loop pattern introduces deliberate points of human oversight into agent-driven workflows, especially when risk, uncertainty, or business impact is high. Instead of fully automating every decision, the system routes certain outputs to a human for review, approval, or adjustment before execution. This may be triggered by low confidence, rule violations, or predefined thresholds such as order size or complexity. The pattern ensures that critical decisions remain accountable while still benefiting from agent efficiency. In practice, it allows organizations to adopt agentic systems safely by combining automation with human judgment where it matters most.

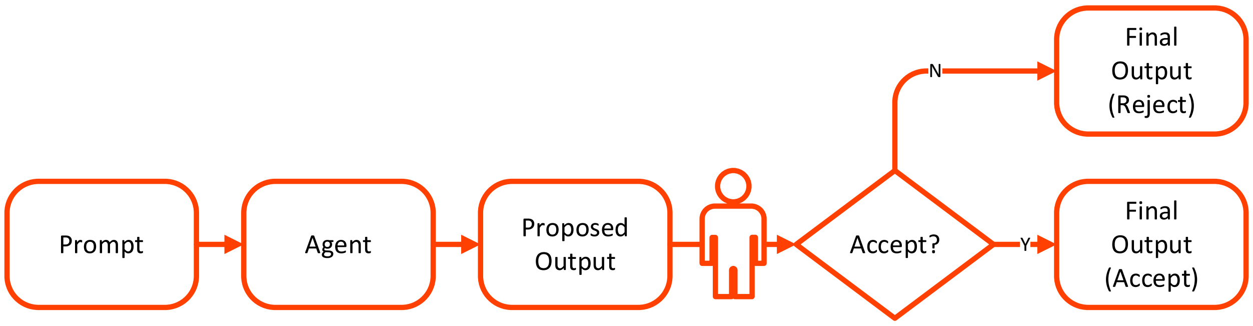

*Figure 10 - Human-In-The-Loop*

### *WHY IT MATTERS*

The human-in-the-loop pattern builds trust in agent-driven systems by introducing oversight where risk is high. In commerce, errors have real consequences, so critical decisions benefit from human review. These checkpoints allow organizations to adopt automation safely and expand it over time as confidence grows. They also preserve accountability by combining the speed of agents with human judgment where it matters most.

<u>Examples</u>

- Large quotes requiring approval
- Edge-case configurations
- Uncertain substitutions

### *COMMERCE INSIGHT*

Agentic commerce is not about removing humans. It is about elevating them. The best systems know when to escalate.

## SO WHAT NOW?

The next step is not to build a full agentic commerce system. It is to start small, with a focused problem where the value is clear and the risk is manageable. Look for a use case that already requires multiple systems and manual effort, such as product substitution, quote generation, or order validation. These are strong candidates because they expose the gaps that agentic patterns are designed to solve.

From there, shift your thinking from endpoints to capabilities. Instead of asking what APIs you have, ask what outcomes your system should deliver. Define a small set of capabilities and map them to clean, deterministic tools. This is the foundation. Without reliable tools and structured data, agents will not perform consistently.

Next, introduce one or two patterns intentionally. Prompt chaining and tool use are often the best starting point because they create immediate value and are easier to control. Add reflection or constraint enforcement as needed to improve reliability. Avoid trying to implement every pattern at once. Progress comes from layering these patterns over time, not deploying them all upfront.

Finally, build with observability and control in mind. Log decisions, track tool usage, and make it easy to inspect how outcomes are produced. This is critical for trust and iteration. Agentic systems are not set-and-forget. They improve through visibility, feedback, and refinement.
The goal is not perfection on day one. The goal is to establish a foundation where agents can operate reliably, then expand their role as confidence grows.

**Scan to learn more about Layer One**
[layerone.us](layerone.us)