SAP PRESS



Agentic AI with SAP®

- Prepare your SAP environment for agentic AI implementation
- Decompose complex business processes into tasks agents can execute
- Develop low-code and pro-code agents with Joule Studio and SAP AI Core

Raghu Banda
Shibaji Chandra

Rheinwerk Publishing

Contents

Notes on Usage 5
Preface 15

1 Agentic AI in the Enterprise 23

1.1 From Transactions to Intent: The Architectural Evolution of SAP 25
1.1.1 The Transactional Era: Monoliths, Transaction Codes, and the Transaction SE80 Paradigm 25
1.1.2 The User Experience Era: Decentralization, SAP Fiori Apps, and OData Gateways 26
1.1.3 The Agentic Era: The Paradigm of Generative Intention and the Agent Fabric 27
1.1.4 Architectural Comparison: Deterministic State Versus Stochastic Flow 29

1.2 The Agentic Shift: The App and Agent Paradigm 30
1.2.1 The Division of Labor: Determinism Versus Probability 30
1.2.2 The Mechanics of Coexistence: How an App and an Agent Collide 31

1.3 Industry Patterns: Art of the Possible Examples 33
1.3.1 Supply Chain Management: The Self-Healing Logistics Network 34
1.3.2 Finance: The Continuous, Zero-Touch Close 34
1.3.3 Spend Management: Autonomous Sourcing and Dynamic Negotiation 35
1.3.4 Human Capital Management: Contextual Talent Orchestration 36
1.3.5 Customer Experience: Proactive Retention and Resolution 36

1.4 Summary 37

2 Foundations of Agentic AI in SAP 39

2.1 The Agentic Loop from Prompt to Action 40
2.1.1 Perception (the Input) 41
2.1.2 Reasoning (the Brain) 42
2.1.3 Planning (the Strategy) 42
2.1.4 Action (the Hands) 43
2.1.5 Reflection (the Quality Check) 43
2.1.6 Walking the Loop: The Worked Example 44

2.2 Automation Versus Agency: A Fundamental Shift 45
2.2.1 The Spectrum: From Scripts to Reasoning 46
2.2.2 The Context Gap: Why Agents Succeed Where Bots Fail 48

2.2.3 The Hybrid Model: Agents as Orchestrators 49
2.2.4 The Decision Matrix: When to Use What? 49

2.3 The SAP Agentic Stack 50
2.3.1 Generative AI Hub: The Central Switchboard 53
2.3.2 SAP Domain Models: The Business Context Engine 56
2.3.3 SAP Knowledge Graph: The Semantic Grounding Layer 58
2.3.4 SAP Business Data Cloud: The Unified Data Foundation 62

2.4 Summary 65

3 SAP Enterprise Architecture for Agentic AI Systems 67

3.1 Shifting Enterprise Architecture Paradigm 68

3.2 The Role of the Architect 69
3.2.1 Augmented Enterprise Architect 70
3.2.2 Different Architect Personas 70
3.2.3 Why Agentic AI Changes the Enterprise Architecture Mandate 72
3.2.4 Expanded Responsibilities for Enterprise Architects in the Autonomous Enterprise 73

3.3 Capability Mapping 76
3.3.1 Business Capability Map 76
3.3.2 Technology Capability Model for AI 78

3.4 Data-Centric Design 82

3.5 AI Agent Hub in SAP LeanIX 85
3.5.1 Benefits 85
3.5.2 AI Agent Discovery 86
3.5.3 AI Agent Governance 86
3.5.4 Model Context Protocol Server for SAP LeanIX 89

3.6 Designing AI Agents for the Enterprise 90
3.6.1 Governing Principles 90
3.6.2 Design Principles 91
3.6.3 Operating Principles 92

3.7 Summary 93

4 Architectural Patterns for Business Processes 95

4.1 Decomposing Business Processes for Agency 96
4.1.1 The Agenticness Assessment Matrix 97
4.1.2 Deconstructing the Order-to-Cash Dispute 100

4.2 The Runtime Spectrum: Embedded, Managed, and Custom Agents 102
4.2.1 The Embedded Paradigm: Joule and Native Agenticness 103
4.2.2 The Managed Agent Runtime (Declarative Design) 105
4.2.3 The Custom Agent Runtime (Code-Based Orchestration) 107
4.2.4 Choosing a Runtime: Embedded, Managed, or Custom 109
4.3 Pattern 1: The Supervisor (Multiagent Orchestration) 110
4.3.1 Joule as the Central Supervisor 111
4.3.2 Designing the Tool Registry (Model Context Protocol) 112
4.3.3 Orchestration Styles for Multiagent Systems 114
4.4 Pattern 2: Human in the Loop 115
4.4.1 Redefining the Approval 116
4.4.2 The Approval Handoff Mechanism: From Routing to Reasoning 117
4.4.3 Designing for Exceptions: The Graceful Failover 119
4.5 Pattern 3: Reflection Loops (ReAct) 120
4.5.1 The ReAct Framework 121
4.5.2 The Architecture of Self-Correction: Moving Beyond Try/Catch 122
4.5.3 Bounding the Loop: The Architecture of Constraint 124
4.6 Summary 126

5 Reference Architecture for Agentic AI in the SAP Landscape 129
5.1 Component Architecture in the SAP Landscape 130
5.1.1 Data Layer 131
5.1.2 AI Layer 132
5.1.3 Decision Layer 133
5.1.4 User Experience Layer 134
5.1.5 Development Layer 134
5.1.6 Security Layer 135
5.2 How the Layers Work Together: An End-to-End Walkthrough 137
5.3 Architectural Principles for the Enterprise Architect 138
5.4 Summary 139

6 Building Low-Code Agents with Joule Studio 141
6.1 Introduction to Joule Studio 142
6.2 Getting Started with Joule Studio 145
6.2.1 Setup 145
6.2.2 Configuration 149

6.2.3 Developing AI Capabilities 154
6.2.4 Developing Joule Skills 154
6.2.5 Developing Joule Agents 159

6.3 Configuring Grounding 164
6.3.1 Configuring Document Grounding 164
6.3.2 Configuring an AI Agent 166

6.4 Testing and Monitoring 167
6.4.1 Testing 167
6.4.2 Deployment 169

6.5 Summary 171

7 Building Pro-Code Agents on Agent Fabric 173

7.1 The Pro-Code Pathway 173

7.2 Agentic Frameworks 175
7.2.1 LangGraph 175
7.2.2 Pydantic AI 190
7.2.3 Comparing LangGraph, Pydantic AI, and n8n 198

7.3 Joule Code-Based Agent 200
7.3.1 Component Architecture 200
7.3.2 Setting Up the Joule Code-Based Agent 202
7.3.3 Understanding the Joule Code-Based Agent Framework 208
7.3.4 Testing and Deployment 211
7.3.5 Observability, Debugging, and Tracing 212

7.4 Integration with Agent-to-Agent 213
7.4.1 Architecture Overview 214
7.4.2 Invoking a Third-Party Agent via Agent-to-Agent 216
7.4.3 Security Considerations for Agent-to-Agent 217

7.5 Agent Harness for Production 221
7.5.1 Scope and Boundary 221
7.5.2 Context Engineering 222
7.5.3 Security Architecture 224
7.5.4 Deployment Architecture 225

7.6 Summary 226

8 Building Agents with Vibe Coding 227

8.1 Joule in the Autonomous Enterprise 228
8.1.1 Joule: From Digital Assistant to Engagement Layer 228
8.1.2 SAP-Managed Joule 230

8.1.3 Architectural Path 231
8.2 The Intent-Driven Workflow 233
8.3 Headless Architecture 233
8.4 Joule Studio 2.0 237
8.5 From Vibe to Pro-Code: A Seamless Graduation 238
8.5.1 One Artifact, Three Paths: Vibe, Low-Code, and Pro-Code 238
8.5.2 Agent Development 239
8.5.3 App Development 241
8.5.4 Workflow/Process Flow Development Using n8n 247
8.5.5 Comparing Low-Code, Pro-Code, and Vibe Coding 254
8.6 Common Pitfalls of Vibe Coding 255
8.7 Summary 256

9 Security, Trust, and Governance 257
9.1 Introduction: Why Autonomous Agents Require New Security Models 258
9.2 The SAP Framework for Responsible AI 259
9.2.1 Pillar 1: AI Ethics and Safety 260
9.2.2 Pillar 2: AI Security and Data Privacy 261
9.2.3 Pillar 3: AI Compliance 261
9.3 The SAP AI Trust Layer 262
9.3.1 Why a Raw Large Language Model Cannot Be Deployed Directly 263
9.3.2 Architectural Positioning 264
9.3.3 Input Filtering 265
9.3.4 Output Filtering 267
9.3.5 From Trust Layer to Agent Hub 268
9.4 Threat Modeling for Autonomous Agents 268
9.4.1 Chain-of-Thought Hijacking 269
9.4.2 Tool Use Vulnerabilities 270
9.4.3 Memory Manipulation 270
9.4.4 How to Implement Threat Modeling in SAP 271
9.5 Enterprise Security Architecture: Identity and Access in the Agentic Era 283
9.5.1 The Privilege Escalation Trap (the God Mode Problem) 284
9.5.2 Principal Propagation: The Identity Lifeline 285
9.5.3 How to Configure Principal Propagation for an Agent Identity 286
9.5.4 Resource Groups: The Isolation Boundary 290
9.5.5 Joule Sovereignty: Mapping Autonomy to the Human 291

9.6 Auditable Reasoning 291
9.6.1 The End of Code-Only Audits 292
9.6.2 Three Audit Artifacts That Didn't Exist Before 292
9.6.3 The Evidence Chain: What the Platform Produces and What You Must Build 293
9.6.4 Why This Matters for the Chief Information Security Officer and the Audit Committee 294
9.6.5 How to Establish Auditable Reasoning Logs for Compliance 294

9.7 Setting Up an Agent Governance Board 302
9.7.1 Mandate and Charter 303
9.7.2 Composition: Four Core Roles 304
9.7.3 The Agent Intake and Review Process 304
9.7.4 Ongoing Oversight Cadence 305
9.7.5 Integration with SAP AI Agent Hub 306
9.7.6 Aggregate Trust Telemetry Dashboard: Weekly Review 311
9.7.7 Override and Intervention Panel: Real Time 313

9.8 Summary 314

10 Operating the Agent Ecosystem 317

10.1 The AgentOps Mandate: Moving from Pilot to Scale 318
10.1.1 The Pilot Purgatory (the 74 Percent Problem) 318
10.1.2 The Fallacy of Outside-In AI 319
10.1.3 The AgentOps Paradigm: Managing Stochastic Systems 320
10.1.4 The Agent-Led Toolchain 322

10.2 The Generative AI Center of Excellence 324
10.2.1 The Mandate: From Shadow AI to Governed Innovation 325
10.2.2 Roles in the Agentic Workforce 327
10.2.3 The Generative AI Center of Excellence as the Transformation Engine 329

10.3 Agent Discovery and the Registry of Intent: SAP LeanIX 331
10.3.1 The Risk of Registry Rot 332
10.3.2 The Agentic Register: Mapping Intent 332
10.3.3 Tool-Calling Lineage and Impact Analysis 335
10.3.4 Continuous Rationalization of the Synthetic Workforce 336

10.4 Technical Lifecycle and Observability: SAP Cloud ALM 338
10.4.1 Agentic DevOps and Continuous Integration/Continuous Delivery 338
10.4.2 Token Metering and FinOps 340
10.4.3 Deep-Loop Observability (OpenTelemetry) 342
10.4.4 The Runtime Kill Switch Protocol 343

10.5 Business Process Monitoring: SAP Signavio 344
10.5.1 Process Mining the Un-User Experience 345
10.5.2 The Key Performance Indicators of Autonomy: Measuring Core Impact 346
10.5.3 Value-Based Governance and the Feedback Loop 349
10.5.4 Validating Agent-Led Transformation 350

10.6 User Adoption and Enablement 351
10.6.1 The Trust Paradox 351
10.6.2 Confidence Scores as the New User Interface 352
10.6.3 From Operator to Orchestrator (the New Human Advantage) 353
10.6.4 Vibe Coding as a Cultural Catalyst 355

10.7 Summary 356

11 The Agentic Roadmap and Future Outlook 359

11.1 The Autonomous Enterprise 360
11.1.1 Why Now: Value Is Shifting to the Agentic Layer 360
11.1.2 SAP's Right to Win 361
11.1.3 The Foundation: SAP Business AI Platform 362
11.1.4 The Execution Paradigm and the Clean Core Prerequisite 363
11.1.5 The Five Autonomous Domains 364
11.1.6 Real-Time Coordination and Built-In Trust 366

11.2 Beyond Joule: The Orchestration Era 367
11.2.1 Joule Work: The Evolution of the Enterprise Workspace 368
11.2.2 Assistants and Agents: Teammates and Executors 369
11.2.3 SAP's Agent Fabric: Orchestrating the Fleet 370
11.2.4 Cognitive Capabilities: Planning, Reflection, and Cross-System Reasoning 370

11.3 Decentralized Ecosystems: Agent to Agent and Model Context Protocol 372
11.3.1 Model Context Protocol: Unifying the Tooling Ecosystem 372
11.3.2 Agent-to-Agent Protocols: Business-to-Business Cross-Boundary Negotiation 373
11.3.3 The Open Partner Agent Ecosystem 374

11.4 SAP and Hyperscalers 375
11.4.1 Model Sovereignty via the Generative AI Hub 376
11.4.2 Zero Data Retention and Collaborative Innovation 377
11.4.3 Cloud Portability and Future Proofing 378

11.5 Conclusion: The Beginning of Better 379
11.5.1 What Changes: The Transformation of Enterprise Operations 379
11.5.2 What Doesn't Change: The Immutable Laws of Enterprise Architecture 379
11.5.3 The Path Forward for the Enterprise Architect 380
11.5.4 Key Takeaways for the Enterprise Architect 380

The Authors 383
Index 385
Service Pages 393
Legal Notes 393

Chapter 2
Foundations of Agentic AI in SAP

This chapter defines the core cognitive architecture of an SAP agent (perception, reasoning, planning, action, and reflection) and maps each concept directly onto the SAP technology stack. By the end, you will understand both what an agent is and the role of the precise SAP machinery that makes probabilistic reasoning enterprise-ready, secure, and governable.

What makes an SAP agent tick? To get started, we need to break down an SAP agent's "mind" to learn how it perceives, reasons, plans, and acts. We will connect these ideas directly to the SAP technology you'll be using, like the generative AI hub, SAP Domain Models, and SAP Knowledge Graph.

In the last chapter, we walked through the evolution of the SAP interface, from the old-school transaction codes of the SAP GUI days to the new, intent-driven conversations with Joule. We made the case that the future of work isn't just about doing things faster; it's about having systems that can manage complex tasks on their own. Now we're moving from the big picture to the nuts and bolts of how these agents actually think.

At its heart, an agent is a system built to tackle complex goals with minimal hand-holding. This is a huge departure from a traditional workflow that just follows a rigid, predefined script. An agent operates in a world of uncertainty. It senses what's going on, figures out the best way forward, uses the tools it has, and, most importantly, learns from the results to do better next time.

For an SAP architect, this completely flips the script. We used to write code to anticipate every possible error or edge case. With agents, our job is to give the system the right tools (APIs, now exposed through Model Context Protocol [MCP]), context (SAP Knowledge Graph), and guardrails (trust and safety layers). Then, we empower the reasoning model to figure out the best path on its own, in the moment. This is the magic behind what SAP calls *Joule agenticness*. Joule is no longer a passive copilot waiting for your next question; it's the engagement layer and the conductor of an autonomous workforce. We're moving beyond simple task automation and into business decision augmentation, where AI helps us manage complexity and navigate the gray areas.

To make this possible, the modern SAP landscape is anchored by Joule Studio 2.0, SAP's unified, AI-first environment for building agents, apps, and workflows running on the SAP-managed Joule Studio runtime, with Joule at the center of it all. Joule can assign tasks to specialized agents, let those agents ask you for clarification, and let agents discover and invoke a shared catalog of tools and skills (standardized through MCP) so that every team

isn't stuck reinventing the same API calls. These agents are organized across SAP's five autonomous domains: Autonomous Finance, Autonomous Spend Management, Autonomous SCM, Autonomous HCM, and Autonomous CX, all extended by Industry AI. In the next few sections, we'll dive into the mechanics of this system, exploring how agents plan, use tools, reflect, and collaborate, and we'll map out the SAP agentic stack that powers it all. The next section will introduce the agentic loop and examine it in the context of SAP at a conceptual level.

2.1 The Agentic Loop from Prompt to Action

To an end user, an agent looks like a chatbot. To an architect, an agent is a loop. This section demystifies the runtime behavior of an agent, explaining that it is not just "predicting the next word" (like a standard large language model [LLM]) but actively solving a problem through a recursive process.

At the heart of every agentic AI system lies a self-sustaining cognitive loop, commonly known as the *agentic loop*. This loop is the cornerstone of intelligent behavior, enabling agents to observe the world, interpret what they see, devise plans, act on those plans, and then reflect on the outcomes. Unlike traditional automation, which executes predefined logic, agentic systems are adaptive, interactive, and purpose-driven.

Figure 2.1 illustrates the cyclic nature of this agentic AI process at a high level.

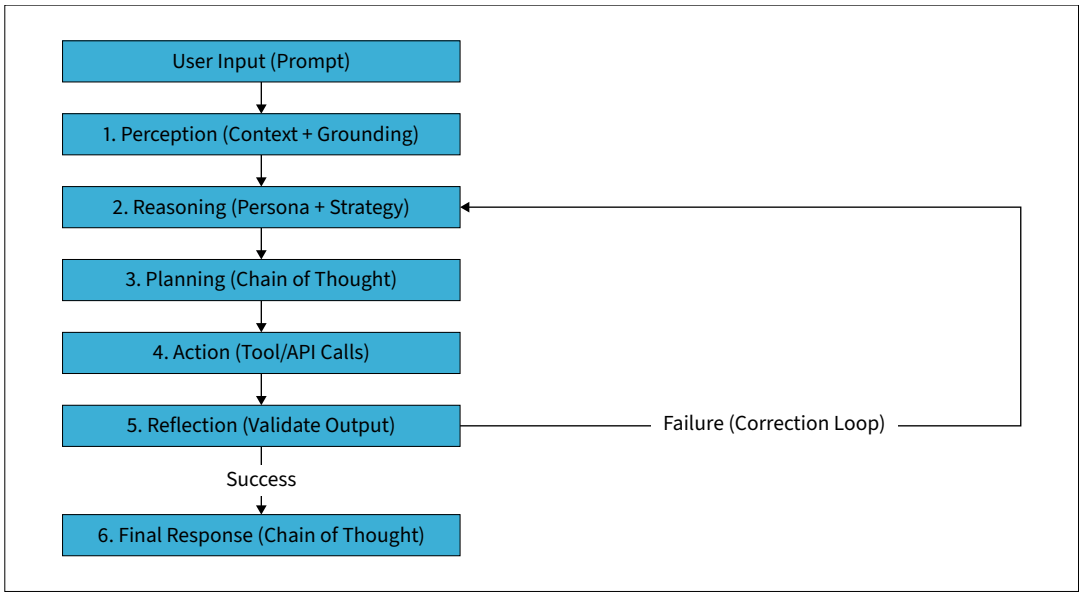


Figure 2.1: Agentic Loop

The agentic loop is best understood as five tightly coupled stages: perception, reasoning, planning, action, and reflection. Each stage serves a distinct function but is tightly coupled

with the others to support continuous feedback, learning, and adaptability. The first four stages drive the agent forward, but it is reflection, the fifth stage, that distinguishes a true agent from a simple automated script.

Reflection is also where safety lives. Because an agent can act, not just display, every cycle passes through a guardrail layer, the agent's immune system, which checks outputs against verified enterprise data before any action is executed. SAP implements this as the AI trust layer, which will be covered in depth in Chapter 9.

Before walking the five stages, it helps to know the five parts an agent is built from. If the loop is the process, these are the components that run it. At the center is the LLM, the reasoning engine, which interprets input, plans next steps, and generates output in natural language or a structured format. Around the LLM sit four supporting elements. *Knowledge* is the contextual data the agent draws on, sourced from structured systems such as SAP HANA and SAP S/4HANA APIs and from unstructured content such as documents and images. *Memory* holds the transient state, the intermediate results that give the agent continuity across a multistep task. *Tools* are the actionable interfaces (APIs, datasets, datastores, process calls) the agent invokes to affect the world. Finally, the *recipe* is the orchestration logic, the guidebook that tells the agent how the LLM, knowledge, and tools should work together. With these five parts in mind, the five stages of the loop become easy to follow.

The agentic loop begins the moment an agent receives a trigger such as a user request, a system event, or a scheduled task. From that trigger, it moves through the five stages discussed previously: perception, reasoning, planning, action, and reflection.

2.1.1 Perception (the Input)

In SAP, input isn't just a text prompt; it is a context object containing the user's role (e.g., procurement manager), the current SAP Fiori app state (e.g., purchase order #1234), and the business intent. In SAP agentic systems, *perception* refers to how the agent receives and interprets signals from the business landscape. These inputs include structured and unstructured data across SAP applications and beyond.

A *perception channel* is any pathway through which an agent receives input from its environment, such as a user's chat prompt, an SAP Event Mesh notification, an incoming email or document, or an API callback. The most common channels in an SAP landscape include the following:

- SAP Business Data Cloud (the unified data foundation, encompassing SAP Datasphere and SAP HANA Cloud)
- Event Mesh (event-driven architectures that notify agents of changes)
- SAP Integration Suite (aggregates data from hybrid or multicloud environments, and now exposes those integrations as MCP-discoverable tools)
- Joule-powered natural language interfaces (interpreting user queries as perception events)

Perception is not just data ingestion. It involves semantic grounding; for example, agents must convert raw inputs into contextual signals using the following:

- SAP Knowledge Graph
- Metadata from SAP Signavio
- Foundational knowledge from SAP AI Core or the generative AI hub

You can think of it this way: If the agent was a human, perception would be like hearing a colleague say "We're low on stock in Region B," and instantly relating that to products, customers, and seasonal demand.

2.1.2 Reasoning (the Brain)

The agent does not jump to an answer. First it consults its system instructions (a persona, the role it is told to adopt) and available tools (skills) to formulate a hypothesis. The persona shapes the tone, priorities, and risk posture, steering how the reasoning model interprets every subsequent input—for example, "To answer this, I first need to check stock levels, then check supplier lead times."

Once they have the information they need, agents engage in reasoning. In this phase, the agent evaluates the situation, interprets intent, and determines options. SAP agents can tap into the following:

- SAP Domain Models (SAP-RPT-1.5 for tabular data and SAP-ABAP-1 for code), alongside leading third-party models accessed through the generative AI hub
- Joule Agents with built-in reasoning chains
- Business rules and policies stored in the business rules engine
- Domain knowledge ontologies modeled through SAP Business AI Platform

Reasoning combines deterministic logic (e.g., inventory thresholds, policy rules) with probabilistic inference (e.g., predicting demand fluctuation). This dual mode enables agents to operate in gray zones when data is incomplete or ambiguous.

Consider a concrete enterprise example. When an overdue payment event fires in SAP S/4HANA Finance, a warehouse agent determines that a delay in delivery is likely due to a supplier disruption and decides to evaluate alternate suppliers before flagging it to a human.

2.1.3 Planning (the Strategy)

During planning, the agent breaks a big goal down into small, manageable steps. You might know this as the *chain of thought* (CoT). The agent essentially creates a to-do list, turning its reasoning into a concrete plan of attack. In SAP, this involves the following:

- Using agentic frameworks such as LangGraph (and now LangChain, LlamaIndex, and AutoGen) within SAP AI Core to map out the steps
- Consulting SAP Signavio to understand the business process ("Is this a procurement step or a fulfillment step?")
- Choosing the right Joule Skills for the job, like Create Purchase Order or Check Credit Limit

The agent determines the best sequence of actions, which systems to call, and whether it can handle the whole process itself or needs to loop in a human for approval. This is where its persona and skills really shine. A procurement persona, for instance, would be equipped with skills for approving requisitions, contacting vendors, and running risk checks.

2.1.4 Action (the Hands)

Finally, the agent acts on its plan. This is where it actually does something, like making an API call to an OData service or pulling specific data from a system. Increasingly, these actions are not hardwired integrations but tools the agent discovers and invokes through MCP, and when a task requires collaboration with another agent, the agent reaches out over the A2A protocol.

Actions are carried out via the following pathways:

- SAP Cloud Application Programming Model APIs and other OData/REST services, exposed as MCP tools through SAP Integration Suite
- SAP Build Process Automation for robotic tasks
- Joule Skill invocations to trigger prebuilt functions

Throughout the action, the agent ensures that everything is secure (using SAP Cloud Identity Services' identity and access management), that its work is explainable (by logging its steps), and that it's getting real-time feedback to know if the action succeeded. For example, an agent might create a service ticket, draft an email to a supplier, and update the enterprise resource planning (ERP) system, all while leaving a clear audit trail in SAP Cloud ALM.

2.1.5 Reflection (the Quality Check)

Many advanced agents have a fifth, crucial step: reflection. After taking an action, the agent steps back and analyzes the result.

It asks itself several questions:

- Did that work?
- Did the user have to step in and correct me?
- Could I have done that more efficiently?
- Was my initial reasoning correct?

This feedback loop is what allows the agent to continuously learn and get smarter over time. In SAP, this could be powered by fine-tuning loops in SAP AI Core or by analyzing user feedback (like a thumbs-up or thumbs-down in Joule). The reasoning traces produced here are not discarded; they become first-class audit artifacts, a topic we return to in Chapter 9.

Unlike a simple API call that just does one thing and forgets it, an agent is stateful. It remembers what it has done and learns from it in a continuous, iterative loop of all five stages.

Note

Architects must recognize that the agentic loop incurs a significant cognitive tax in terms of latency and compute, making it unsuitable for every process. Consequently, this recursive pattern should be reserved strictly for ambiguous problems requiring reasoning. Deterministic automation remains the standard for high-volume, repetitive tasks.

2.1.6 Walking the Loop: The Worked Example

Abstract stages are easier to grasp when grounded in a concrete scenario. Consider an accounts receivable collections agent operating within the Autonomous Finance domain. A mid-tier customer, ACME GmbH, has just missed a $48,000 invoice payment by nine days. Let's trace all five stages of the loop as the agent handles this single event:

1. **Perception**
   The agent is triggered not by a human, but by an Event Mesh notification, such as an overdue-payment event fired by SAP S/4HANA Finance. The context object it receives carries the invoice number, the customer ID, the amount, the number of days overdue, and the agent's own persona (collections specialist, EMEA). It doesn't yet know whether this customer is a chronic late payer or the situation is a one-off; perception simply establishes that something happened that warrants reasoning.
2. **Reasoning**
   The agent consults SAP Knowledge Graph to enrich the bare event into a situation. It learns that ACME GmbH has paid on time for 23 consecutive months, that it is flagged as a strategic account, and that a support ticket was opened by ACME four days ago regarding a disputed delivery. Combining deterministic policy ("strategic accounts are not autoescalated to dunning level 2") with probabilistic inference ("a recent support ticket plus a first-ever late payment suggests the delay is dispute-related, not financial distress"), the agent forms a hypothesis: This is not a collections problem but a dispute problem wearing a collections costume.
3. **Planning**
   The agent decomposes its goal ("recover the $48,000 without damaging a strategic relationship") into an ordered plan: (1) Pull the linked support ticket to confirm the dispute;

(2) check whether a credit memo is already in flight; (3) if the dispute is valid and unresolved, draft a conciliatory outreach to the account manager rather than a dunning letter; and (4) hold the automated dunning run for this specific invoice. It selects the skills it will need for each step and decides that step 3 will require human review before anything is sent externally.

4. **Action**
   The agent executes. It invokes an SAP Knowledge Graph traversal to confirm the ticket-to-invoice linkage, calls an MCP-exposed SAP S/4HANA tool to verify no credit memo exists yet, and uses an A2A handoff to ask the dispute resolution agent whether the delivery discrepancy has been substantiated. Each action leaves a reasoning trace in SAP Cloud ALM. The agent then drafts but does not send an email to the account manager and places a hold flag on the dunning run for the invoice.
5. **Reflection**
   Before closing the loop, the agent evaluates its own work. Did SAP Knowledge Graph confirm the linkage? Yes. Did the dispute resolution agent substantiate the discrepancy? Partially; the delivery shortfall is confirmed, but the euro value is still being assessed. The agent reflects that its confidence in the exact credit amount is therefore low. Rather than proposing a specific figure, it routes the drafted outreach to the human account manager with an explicit note: "Dispute confirmed, amount pending; recommend we pause collections and let me re-evaluate once the dispute resolution agent finalizes the credit value." This reflection step is precisely what separates an agent from a script: A script would have sent a dunning letter on day nine, but the agent recognized the limits of its own certainty and escalated gracefully.

This single example illustrates why the loop matters. At no point did a developer hardcode the "if recent support ticket exists, suppress dunning" branch. The agent reasoned its way there from context. That is the difference between automation and agency—and the subject of the next section.

2.2 Automation Versus Agency: A Fundamental Shift

To really grasp why the agentic loop is such a game changer, we need to compare it to the method by which we've been doing things for the last decade: deterministic automation.

For years, *automation* in SAP referred to tools like traditional robotic process automation (RPA) and workflows. These tools are great for what they do, but they're rigid by design. They run on simple if-then logic—for example:

- If an invoice is less than $500, then automatically approve it.
- If you see a `Vendor_ID` field, then copy its value to the `Payee` field.

This is a deterministic architecture. It's fast, cheap, and completely predictable, but it's also brittle. The moment a vendor changes their invoice format, the RPA bot breaks. If a process

requires judgment, like deciding if a shipping delay is acceptable because of a hurricane, the workflow grinds to a halt and waits for a human.

This is where agentic AI, or agency, comes into play! Agentic AI introduces a probabilistic architecture. It operates on goal-seeking logic instead—for example:

- Goal: "Make sure the production line doesn't stop."
- Constraint: "Don't spend more than $10,000 to fix it."

You don't give the agent a script; you give it an objective and the tools to achieve it. The agent must figure out the "how" on its own. This shift from hardcoding steps to defining goals is what lets us move from just automating repetitive tasks to augmenting complex business decisions. It answers a common question we hear from SAP teams: "We already have RPA, so why do we need agents?"

In Table 2.1, we differentiate between traditional automation and agentic AI at a conceptual level. This should give you a basic idea of the differences before we dive into more detailed analysis.

Feature	Traditional Automation (RPA)	Agentic AI
Trigger	Explicit rule ("If X, then Y")	High-level goal ("Resolve the delay")
Logic	Hardcoded workflows	Reasoning via LLM
Data handling	Structured inputs only	Unstructured and structured (docs plus tables)
Resilience	Breaks on unexpected error	Self-corrects/retries (reflection)
Role	The hands (doer)	The brain (orchestrator)

Table 2.1: Automation Versus Agency

To use a simple, practical example, a bot that automatically approves any purchase requisition under $500 is an *automation*. It follows a simple, hardcoded rule. With *agency*, imagine that an agent sees a $5,000 requisition. It notices the vendor is new, so it uses one of its skills to check the vendor's credit risk score. It uses another skill to compare the price against historical averages. Finally, it drafts a summary with a recommendation for the human approver.

The following sections will dive deeper into this distinction.

2.2.1 The Spectrum: From Scripts to Reasoning

To architect effectively, we must stop viewing automation and AI as binary opposites. Instead, they exist on a spectrum of autonomy. It's a mistake to see automation and AI as a simple on/off switch. It's better to think of them as a spectrum of autonomy, as follows:

- Level 1: The transactional script (the hands)
  - *Technology:* ABAP reports, BAPIs, scripting
  - *Behavior:* A user clicks a button, and the system does exactly what it was told to do. It's purely reactive.
  - *Role:* Passive execution.
- Level 2: The process automator (the spine)
  - *Technology:* SAP Build Process Automation (including SAP Intelligent Robotic Process Automation), SAP Signavio Process Governance
  - *Behavior:* The system follows a predefined flowchart. It can handle simple branching logic (if/else), but it can't handle ambiguity. It moves data around without understanding what it means.
  - *Role:* Rigid orchestration.
- Level 3: The agentic orchestrator (the brain)
  - *Technology:* Joule, SAP Domain Models, AI agents built in Joule Studio 2.0
  - *Behavior:* The system understands a situation, uses its knowledge to reason about the best path forward, and builds a plan on the fly. It doesn't just see data; it understands meaning.
  - *Role:* Adaptive decision-making. Within this level, agents themselves range in complexity, from a single model call to an agent equipped with tools and memory to full multiagent orchestration, which Chapter 4 explores in depth.

Figure 2.2 illustrates these three levels.

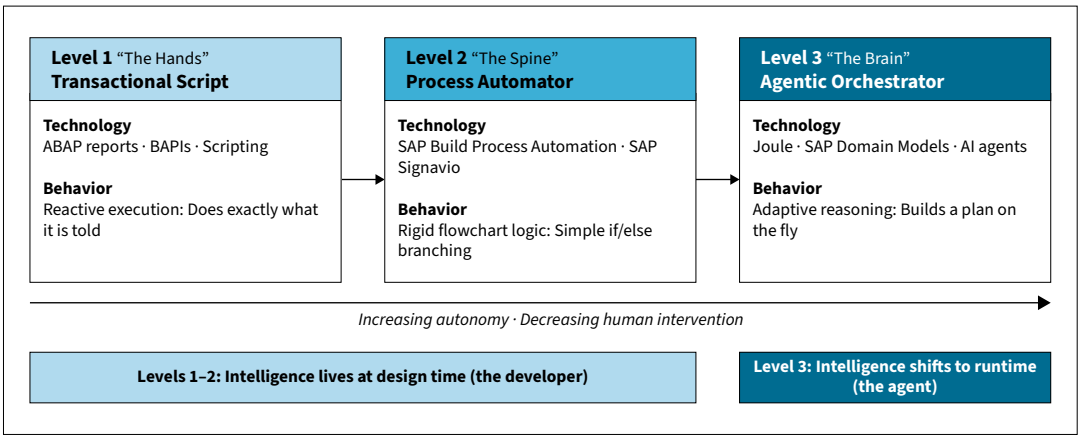


Figure 2.2: Spectrum of Autonomy

In Level 2, the intelligence is all at design time; it's the developer who must think through every possible scenario. In Level 3, the intelligence moves to the runtime; it's the agent reasoning through the specific context of the problem at hand. This is where we need to understand the context and how agents succeed.

2.2.2 The Context Gap: Why Agents Succeed Where Bots Fail

The main reason we need agents is to cross the context gap. In the real world of SAP, data is never perfect: Fields are missing, information is buried in unstructured text, and business rules always have unwritten exceptions.

Consider this scenario: A customer sends an email that says, "Please cancel our order for the blue widgets. We found them cheaper somewhere else." The automation approach (via RPA) would look something like this:

1. The bot scans the email subject, looking for a perfectly formatted order ID.
2. If the ID is missing, the bot fails and creates an error ticket.
3. If it finds the ID, it triggers a hardcoded *cancel order* transaction.
4. *Result:* The bot fails to understand *why* the order was canceled ("found them cheaper"). It misses a critical piece of competitive intelligence. It doesn't know to offer a discount to save the sale because that logic was never programmed in.

On the other hand, the agentic approach (via Joule) would look something like this:

1. **Perception**
   The agent reads the email and understands both the intent (cancel order) and the reason (price sensitivity).
2. **Reasoning**
   It checks its knowledge graph. "This is a gold-tier customer with a high risk of churn. Our policy allows for a 5 percent discretionary discount to retain key accounts."
3. **Planning**
   It creates a new plan: "Don't cancel the order just yet. First, draft a reply to the sales rep suggesting we offer a 5 percent discount. If the customer rejects it, then proceed with the cancellation."
4. **Action**
   It drafts the email for the sales rep to review and send.
5. **Reflection**
   Before it stands down, the agent checks its own work. Is the order still open, or has it already been picked and shipped? Is a 5 percent discount within the discretionary limit policy allowed for a gold-tier account? Does the draft name the right order and the right contact? Because a retention offer commits a revenue concession, the agent does not send it. It routes the draft to the sales rep with its reasoning attached, logs the stated reason for cancellation, includes competitor pricing as a competitive signal for the account team, and flags anything it could not verify for human review.

What is the difference here? The bot executed a task (cancel order). The agent managed an outcome (customer retention). This ability to understand unstructured context and apply what feels like business intuition is the core of this architectural shift. So here we are getting into a hybrid model concept where the agent managed the outcome by leveraging an automation task.

2.2.3 The Hybrid Model: Agents as Orchestrators

A common myth is that agents will completely replace RPA. That's not true. Agents are computationally expensive and can have higher latency. RPA bots are cheap and fast for simple, repetitive work.

The best architecture for an intelligent enterprise is a hybrid one. The agent is the brain: It handles the user interaction, the ambiguity, and the decision-making. The automation is the hands: Once the agent decides what needs to be done, it triggers a simple RPA bot or an API to do the heavy lifting efficiently.

Now let's look at how this is handled in the context of SAP, with the Joule plus SAP Build pattern, which is how SAP's reference architecture is designed:

1. Joule (agent): Talks to the user to understand what they need. "Okay, you want to onboard a new supplier based in Germany."
2. Joule (agent): Validates the data against SAP Domain Models and uses its intelligence to validate the data. "This VAT number looks like a valid German format."
3. SAP Build (automation): Joule triggers a bot to do the grunt work: Create the records in SAP S/4HANA, upload documents to SAP Ariba, and send out the standard confirmation emails.
4. Joule (agent): Monitors the bot's progress and reports back to the user. "All done. Your new supplier, ID 123, has been created."

This flow from human to agent to automation to system lets you strike the right balance among intelligence, speed, and cost. Next, let's discuss how to make the right decisions.

2.2.4 The Decision Matrix: When to Use What?

For the SAP architect, the challenge is choosing the right tool. Not everything needs an agent. Table 2.2 provides a decision matrix to help you determine when to consider traditional automation versus agentic AI. Section 2.3 will provide more details in the context of SAP.

Feature	Traditional Automation (RPA/Workflow)	Agentic AI (Joule and SAP AI Core)
Input data	Structured (forms, tables, API payloads)	Unstructured (emails, PDFs, chat, images)
Process logic	Explicit rules (If X > 5, then Y)	Implicit reasoning (Based on policy, what is best?)
Handling errors	Stops and alerts admin	Reflects, retries, and self-corrects

Table 2.2: Decision Matrix

Feature	Traditional Automation (RPA/Workflow)	Agentic AI (Joule and SAP AI Core)
Development	Low-code flowcharts	Prompt engineering and context grounding
Best use case	High-volume, repetitive, stable processes (e.g., invoice entry)	Low-volume, high-complexity, variable processes (e.g., exception handling)
Cost	Low (compute)	High (tokens/inference)

Table 2.2: Decision Matrix (Cont.)

Now that you understand what an agent is (Section 2.1) and how it differs from automation (Section 2.2), let's examine specific SAP machinery, such as the hub, the graph, and the models, that makes this probabilistic reasoning enterprise-ready.

2.3 The SAP Agentic Stack

This is the core technical section. It details the components within SAP Business AI Platform that enable the agentic loop described in Section 2.1.

In the last section, we drew a line between simple automation (just doing tasks) and real agency (managing goals). Now we need to look at the machinery that makes agency possible. A common mistake we see enterprise architects make is thinking that building an agent is as simple as connecting an LLM to a database. The truth is that a raw LLM is nowhere near ready for the enterprise. It has no business context, it can't make sense of complex SAP schemas, and without a layer of control, it's a massive security risk.

To tackle this, SAP gives us the agentic stack, delivered through SAP Business AI Platform. At SAPPHIRE 2026, SAP unified what were previously three separate offerings—SAP Business Data Cloud, SAP Business Technology Platform, and the broader business technology management capabilities—into a single SAP Business AI Platform organized around three pillars: build, contextualize and reason, and govern. You can think of this platform as the operating system for an intelligent enterprise. It creates the necessary layers of abstraction to keep the fast-moving world of AI models separate from your stable, core business processes. It's designed from the ground up to solve the three big challenges of using agents at scale: reasoning, context, and trust.

In the following sections, we will walk through the components of the agentic stack, unpacking each one of them.

Mapped onto the agentic loop, the stack has four conceptual layers that work together every time an agent does anything:

- **The reasoning layer (the brain)**
  At the center of it all is the ability to reason—to make plans, break down tasks, and come up with answers. Through the generative AI hub, the stack gives you model-agnostic, managed access to leading frontier models (Claude, Mistral, Cohere, and others) for general-purpose thinking. At the same time, SAP Domain Models such as SAP-RPT-1.5 for tabular data and SAP-ABAP-1 for code provide a kind of specialized business intuition, trained to understand the complex ERP language, tables, and ABAP code that generic models just don't get. This layer maps to the *build* and *contextualize and reason* pillars of SAP Business AI Platform.
- **The context layer (the grounding)**
  Reasoning without context is just hallucination. An agent can't make a smart decision about a supply chain problem if it doesn't understand the relationships among a vendor, a material, and a plant. This layer is powered by SAP Knowledge Graph, which now spans 7.3 million data fields across the SAP ecosystem and SAP Business Data Cloud. It makes sure every decision is grounded in the live, real-world data of your business. This enables a technique called *graph retrieval-augmented generation* (graph RAG), which is worlds better than a simple text search.
- **The tool layer (the hands)**
  Agents need to interact with the world around them. Therefore, an agent needs to do things in the real world. In the post-SAPPHIRE architecture, this is addressed by the tool layer, which is standardized around two open protocols. Through MCP, agents discover and invoke tools through a shared, standardized way to expose enterprise APIs (like Create Purchase Order) as discoverable capabilities, delivered through the MCP hub and MCP gateway in SAP Integration Suite (general availability in Q2 2026). The A2A protocol is how agents collaborate with one another, including with partner and third-party agents. Above these protocols sits an action/orchestration layer that handles the nitty-gritty of executing tasks, remembering things (persistent memory is now native to the Joule Studio runtime), and keeping track of state. SAP AI Agent Hub serves as the registry and command center for all these agents and tools.

  Because MCP and A2A are foundational to everything that follows in this book, it is worth pausing to understand why two protocols are needed rather than one. The distinction is the difference between using a tool and delegating to a colleague. When an agent needs a deterministic capability such as looking up an invoice, posting a journal entry, or checking inventory, it uses MCP. MCP treats the capability as a function call: The agent discovers what the tool does, what parameters it needs, and what it returns, all without any custom, point-to-point integration code. The tool itself has no memory and no agency; it executes and returns. This is ideal for the thousands of enterprise APIs that an agent must reach, which is exactly why SAP exposes them en masse through the MCP hub and gateway rather than asking each agent team to wire up its own connections.

  A2A solves a fundamentally different problem: What happens when the thing an agent needs is not a static tool but another reasoning agent? When our Collections Agent

needed to know whether a delivery dispute was substantiated, it could not simply call a function; that question requires another agent to perceive, reason, and plan on its own. A2A handles this stateful, multiturn collaboration. The two agents exchange messages as peers, each keeping its own internal logic, tools, and memory private (a black box collaboration), discovering each other through standardized agent cards, and progressing a shared task through states like submitted, working, input-required, and completed. Crucially, A2A is vendor-neutral and governed as an open specification, which is what allows an SAP agent to collaborate with a partner's agent or a third-party agent from Microsoft, Google, or ServiceNow without either side exposing its proprietary internals. In short, MCP connects an agent to a tool, while A2A connects an agent to another agent. Master that distinction and the rest of the SAP agentic architecture becomes far easier to grasp.

- **The trust layer (the immune system)**
  Wrapped around the entire stack is the AI trust layer, which anchors the govern pillar of SAP Business AI Platform. Because these agents are allowed to act, the cost of a mistake is high. This layer is the enforcer. It does everything from anonymizing data such as personally identifiable information (PII) to spotting hallucinations. It's what makes sure the "best guess" nature of AI doesn't violate the black-and-white compliance rules of your business. SAP AI Agent Hub surfaces this enforcement across the entire fleet of agents, more than 55,000 of which are now managed across customer landscapes.

The four technical layers described here show how an agent works. SAP organizes what agents do around five autonomous domains, a framing introduced at SAPPHIRE 2026 that recurs throughout the rest of this book. Each domain represents an end-to-end slice of the business in which assistants and agents execute work across functions rather than within a single transaction: Autonomous Finance (the order-to-cash and record-to-report worlds), Autonomous Spend Management (source-to-pay and procurement), Autonomous Supply Chain Management (plan, make, deliver), Autonomous HCM (hire-to-retire), and Autonomous CX (lead-to-cash and service). A sixth dimension, Industry AI, extends each domain with vertical process knowledge, data models, and regulatory logic drawn from SAP's decades of coinnovation across 25-plus industries. For the architect, the domains matter because they tell you where in the business a given agent lives, while the four-layer stack tells you how that agent is built. The Collections Agent in our earlier example is an Autonomous Finance agent (the where) constructed from the reasoning, context, tool, and trust layers (the how). Keeping these two axes, the business domain and technical layer, clearly separated is one of the most useful mental habits an augmented enterprise architect can develop.

For an enterprise architect, this stack changes the game from application integration to intelligence integration: a big shift! You're no longer just connecting systems and piping data back and forth; you're orchestrating the flow of intent. The agentic stack standardizes how that intent is handled, ensuring that whether you're building a custom agent in Joule Studio 2.0 or just extending a prebuilt one in Joule, the fundamentals, such as safety, connectivity, and grounding, are always the same.

2.3.1 Generative AI Hub: The Central Switchboard

In the agentic stack, if SAP Domain Models are the specialized business brain, then the generative AI hub is the central switchboard. It's a single, secure gateway that connects all your SAP applications and agents to the wider world of LLMs.

For architects, the generative AI hub solves two huge headaches: fragmentation and fear. The AI market is moving at a dizzying pace; new models from OpenAI, Google, Anthropic, and Mistral pop up seemingly every week. Hardcoding an agent for a specific provider's API is a strategic mistake as it creates vendor lock-in, complicates security, and makes it impossible to switch to a better or cheaper model when one comes along.

Figure 2.3 provides details about the generative AI hub, which is utilized as an AI-powered extension. We'll discuss the components involved in the generative AI hub in detail from an SAP context.

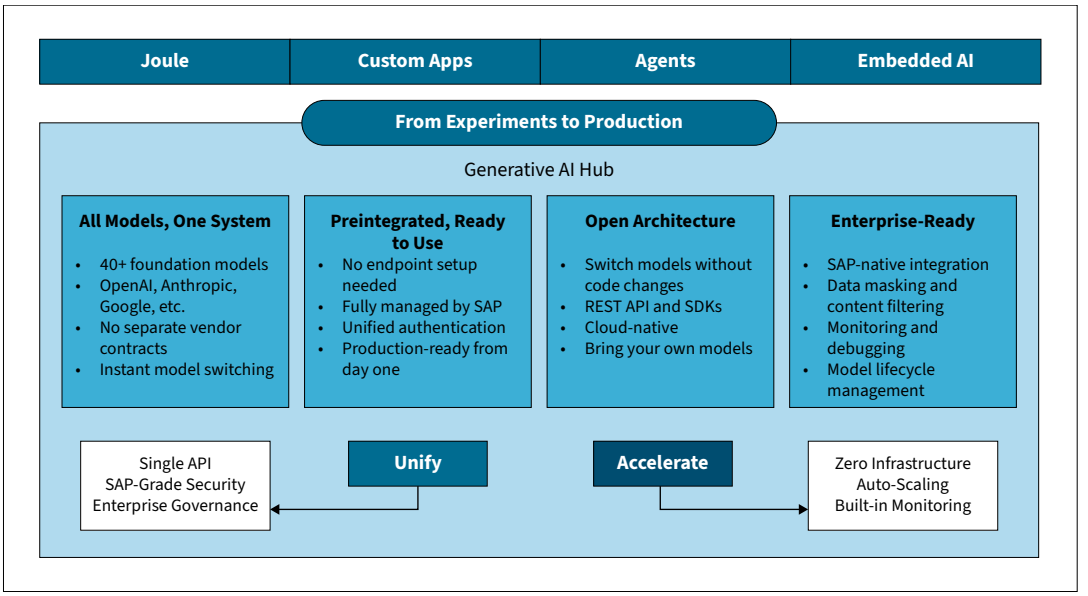


Figure 2.3: Generative AI Hub

The generative AI hub acts as an abstraction layer, sitting between your applications and the raw model providers. It gives you three critical things: model orchestration, enterprise-grade security, and a solid developer workflow.

Now let's dive a bit deeper into the various aspects of the generative AI hub:

- **The multimodel strategy: The model router**
  The core idea of the hub is to give you access without locking you in. It provides instant, managed access to a curated list of the world's best models, covering a huge portion of the market. Instead of juggling separate contracts, API keys, and billing with Azure

OpenAI, Google Vertex AI, and AWS Bedrock, you manage a single connection to the generative AI hub. The hub creates a unified API through SAP AI Core that standardizes how you talk to any model.

It's best to follow a philosophy of choosing the right model for the right task: In an agent-based world, you don't need a sledgehammer for every nut. The hub lets you use a tiered strategy:

- **The heavy lifters (reasoning and planning)**
  For really complex jobs such as analyzing a supply chain disruption or writing ABAP code, agents can call on high-IQ models such as GPT-4o or Gemini 1.5 Pro. These have the massive context windows and logical power needed for the reasoning phase of the agent loop.
- **The speed demons (classification and summarization)**
  For simpler tasks such as figuring out if an email is a complaint or an inquiry or just summarizing meeting notes, agents can be routed to faster, cheaper models such as GPT-4o mini, Claude 3 Haiku, or Gemini 1.5 Flash.
- **The sovereign options**
  For European customers, public-sector entities, or government agencies with strict data residency rules, the hub offers models such as Mistral Large, which can be deployed with tighter regional controls.

The architectural benefit here is huge as this abstraction allows for hot swapping. If a new model comes out that's 50 percent cheaper and 20 percent faster, the Center of Excellence (CoE) can simply update the agent's configuration in the hub to point to the new model, all without touching a single line of the agent's code.

- **The security shield: Zero data retention**
  The biggest barrier to adopting AI like this is the fear of data leaks. Chief information officers are right to worry that their company's private data, sent to an LLM, might be stored or used to train the provider's next model. The generative AI hub is designed to eliminate this risk with a proxy architecture backed by strict legal and technical safeguards and rock-solid contractual guarantees:
  - **Zero data retention policy**
    SAP has specific enterprise agreements with hyperscalers like Microsoft, Google, and Amazon. When your agent sends a prompt through the hub, the provider is contractually and technically forbidden from storing your data or using it for training. Your data only exists in the model's memory for the milliseconds it takes to generate a response, and then it is discarded.
  - **The proxy data flow**
    When an agent needs enterprise data held behind the firewall, it does not connect to the source directly. The request travels through a secure proxy, the cloud connector

and destination layer; this authenticates the call, enforces the acting user's authorizations, and returns only permitted data. The flow works as follows:

- *Request:* The agent sends a prompt to the hub.
- *Sanitization:* The hub (via the AI trust layer) scans the prompt for PII and applies masking (e.g., replacing John Doe with PERSON_1).
- *Transmission:* The masked prompt is sent to the model provider (e.g., Azure OpenAI) via an encrypted tunnel.
- *Generation:* The model generates the response using the masked data.
- *Rehydration:* The hub receives the response, unmasks the data (restoring John Doe), and returns it to the agent.

This architecture ensures that the raw model provider never sees the unmasked PII, and the customer retains full sovereignty over their data lineage. Basically, the process ensures the model provider never gets access to your sensitive data, and you maintain complete control over your data's journey.

- **The developer workbench: From playground to production**
  Building agents takes more than an API call; it takes a proper workflow. The generative AI hub includes tools to streamline the whole build-test-deploy cycle for your developers and prompt engineers. Let's look at what it takes for that to happen:
  - **The playground**
    In a graphical user interface (GUI), developers can experiment with and test different models side by side. They can try prompts against GPT-4, Gemini, and many other models simultaneously to compare latency, accuracy, and cost before making a choice.
  - **Prompt management**
    In agentic AI, a prompt is effectively code. It defines the agent's persona, constraints, rules, and logic. The hub includes a prompt registry that allows teams to treat prompts as software assets. These prompts can be versioned, tagged, and managed centrally. This separates the prompt from the application code, which means, for example, that a business analyst can tweak an agent's tone of voice in the registry without needing a developer to redeploy the whole app.
  - **Software development kit integration (SAP Cloud Application Programming Model and Python)**
    For pro-code development, the hub is fully integrated into the SAP Cloud Application Programming Model and offers a Python software development kit (SDK), letting developers use standard paradigms while authenticating securely against SAP Business AI Platform.
- **Operational governance: The meter**
  Agents can get expensive very fast. An agent that gets stuck in a loop could burn through thousands of tokens a minute. The generative AI hub acts as the central metering point

for the whole enterprise. It provides detailed telemetry on token consumption, broken down by cost center or application. This lets the AI CoE set up chargeback models so that departments using AI (like HR or finance) pay for the resources their agents consume. It also lets you set rate limits to prevent a runaway agent from blowing your entire monthly AI budget in an afternoon.

2.3.2 SAP Domain Models: The Business Context Engine

Although the generative AI hub provides access to general-purpose intelligence—that is, brilliant generalist AI (models trained on the open internet like Wikipedia and Reddit)—enterprise agents often fail when relying solely on general knowledge. A generic LLM is like a brilliant creative writer: It can write a poem about supply chains, but it is not a forensic accountant. It doesn't understand the specific schema of an SAP S/4HANA sales order, it struggles with complex arithmetic, and it lacks the business intuition to detect a subtle anomaly in a general ledger.

Figure 2.4 illustrates the SAP Domain Models approach and the various components involved. This approach is based on structured data, unlike other LLMs that are predominantly based on unstructured data.

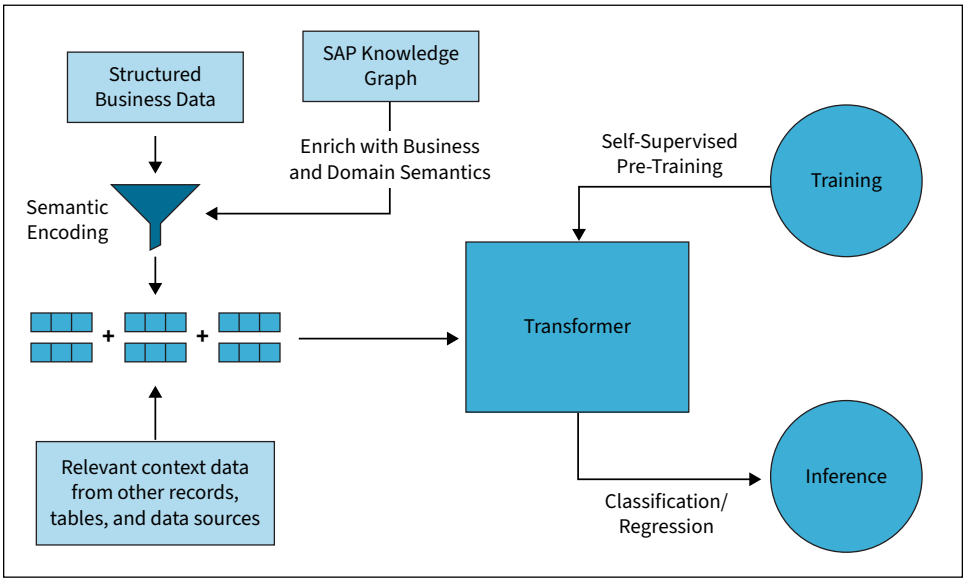


Figure 2.4: SAP Domain Models Approach

SAP Domain Models are proprietary models built by SAP, trained on enormous, anonymized datasets of SAP business data, code, and process flows. For the enterprise architect, SAP Domain Models represent a shift from broad intelligence (knowing a little about everything) to deep domain intelligence (knowing everything about SAP business processes).

Let's now dive into aspects of SAP Domain Models:

- **The internet logic versus business logic gap**
  The primary reason architects cannot rely solely on generic models such as GPT-4 or Claude for every task is what we might call the *hallucination of logic*, and it surfaces in three ways. The first is semantic: asked to "find the customer," a generic model searches for the word "customer" and does not know that in SAP S/4HANA a customer is modeled as a business partner carrying a specific role. The second is mathematical: LLMs are probabilistic token predictors rather than calculators, so they struggle with regression tasks such as predicting exactly how many days a shipment will slip across fifty variables. The third is code: Generic models routinely invent SAP transaction codes or reference BAPIs that have not existed since 2010.

  SAP Domain Models bridge this gap. They are trained on the fundamental "physics" of the enterprise. They understand that a goods receipt must strictly follow a purchase order, and that a credit limit is a hard constraint, not a suggestion.

- **The two classes of SAP Domain Models**
  SAP's strategy focuses on two specific areas in which generic models struggle—structured data and proprietary code:

  - **SAP-RPT-1.5 (relational pretrained transformer) for tabular data (often referred to as SAP-RPT)**
    Most business data is not text; it is structured, tabular data (rows and columns in SAP HANA). Generic LLMs are notoriously bad at analyzing spreadsheet-like data because they treat numbers as text strings.

    Unlike traditional narrow AI (like a random forest model), which requires you to train a specific model for every single use case, SAP-RPT-1.5 is pretrained on millions of anonymized table structures. It understands the concept of an invoice, a date, and a currency, and it ships in small and large versions that balance speed and accuracy.

    A key capability is the missing value imputation. If an agent receives an invoice with a missing payment terms field, a generic LLM would guess or hallucinate. SAP-RPT-1.5 looks at the context (vendor ID, amount, region) and predicts the correct payment terms with high accuracy based on historical patterns of similar vendors. Another key capability is regression and classification. These models excel at answering questions like "What is the likelihood this specific supplier will deliver late?" or "Is this general ledger entry anomalous?"

  - **SAP-ABAP-1 (the ABAP specialist)**
    Agents are increasingly asked to take on developer roles, to extend the ERP system or fix errors, and here generic coding models fall short. Tools like GitHub Copilot excel at Python or JavaScript but often generate syntactically incorrect or insecure ABAP, hallucinating deprecated statements or failing to honor the ABAP RESTful application programming model that SAP S/4HANA Cloud requires. SAP-ABAP-1 solves this because it is trained on SAP's own vast internal repository of rigorously tested ABAP.

The result is that when an agent uses this model to generate an extension, the code is structurally sound, secure by design, and optimized for the SAP HANA database, understanding the nuances of core data services views in a way no external model can.

- **The strategic advantage: The cold start solution**
  One of the biggest hurdles in enterprise AI is the sheer volume of data traditionally required. Consider a company launching a new product line that wants an AI to predict demand. The traditional approach tells them to come back once they have three years of sales history, roughly 10,000 records, before a model can be trained. The SAP Domain Models approach removes that wait: Because the model is pretrained on global patterns, it supports few-shot learning and can make accurate predictions from as few as 50 or 100 records, using its general knowledge of how products sell to fill the gaps where the customer's own data is thin. This solves the cold start problem, letting architects deploy agents into new business units immediately rather than waiting years to accumulate training data.
- **Privacy and the clean core**
  A critical differentiator for SAP Domain Models is how they handle privacy compared with fine-tuning an external model: You do not send your data to be mixed into the base model. When the models are used through SAP AI Core, your data serves only as context for inference and never retrains the base model, so there is no data leakage. The models also reinforce clean core discipline; an agent powered by SAP-ABAP-1 suggests extensions built on public APIs and allowlisted objects, actively steering developers away from the "dirty" modifications that break future upgrades.
- **Deployment: embedded versus side by side**
  For the architect, these models are consumed in two ways. In the embedded model, you do not build the agent so much as switch it on; SAP embeds SAP Domain Models directly into its software-as-a-service applications, like SAP Cash Application in SAP S/4HANA Finance, for instance, in which the agent runs in the background, matching payments to invoices autonomously. In the side-by-side model, the same models are exposed via APIs in SAP AI Core for custom work; you might build an agent in Joule Studio 2.0 that calls SAP-RPT-1.5 to predict customer-churn risk, then hand that result to a frontier model to summarize into an email.

2.3.3 SAP Knowledge Graph: The Semantic Grounding Layer

A reasoning model is only as reliable as the context it is given. Generative AI, left to its own devices, will confidently fabricate an answer when the underlying data is thin or ungrounded, which in an enterprise is not a quirk but a liability. The discipline of preventing this is context engineering: deliberately feeding the agent verified, high-quality enterprise data so that its perception and reasoning stages are anchored in fact rather than probability. SAP delivers context engineering through two complementary layers: SAP Knowledge

Graph, which supplies the semantic meaning of enterprise data, and SAP Business Data Cloud, which supplies the unified data itself.

If you think of the generative AI hub as the brain and its models as skills, then SAP Knowledge Graph is the memory. More importantly, it's the conscience that keeps the brain honest. Spanning 7.3 million data fields across the SAP ecosystem, it's the connective tissue that gives agents real operational context rather than raw data.

For decades, enterprise data has been locked away in thousands of tables in SAP ERP with names only an engineer could love, like tables `VBAK`, `MARA`, and `BSEG`. These tables are all connected, but figuring out how has always required an ABAP expert or a SQL wizard who can navigate the dizzying web of foreign keys. To an AI agent, that world is a complete black box. It can't just look at the data and "see" that a sales order is blocked because the customer is in a region currently under sanctions.

Figure 2.5 provides a glimpse of SAP Knowledge Graph evolution across the various applications in SAP systems.

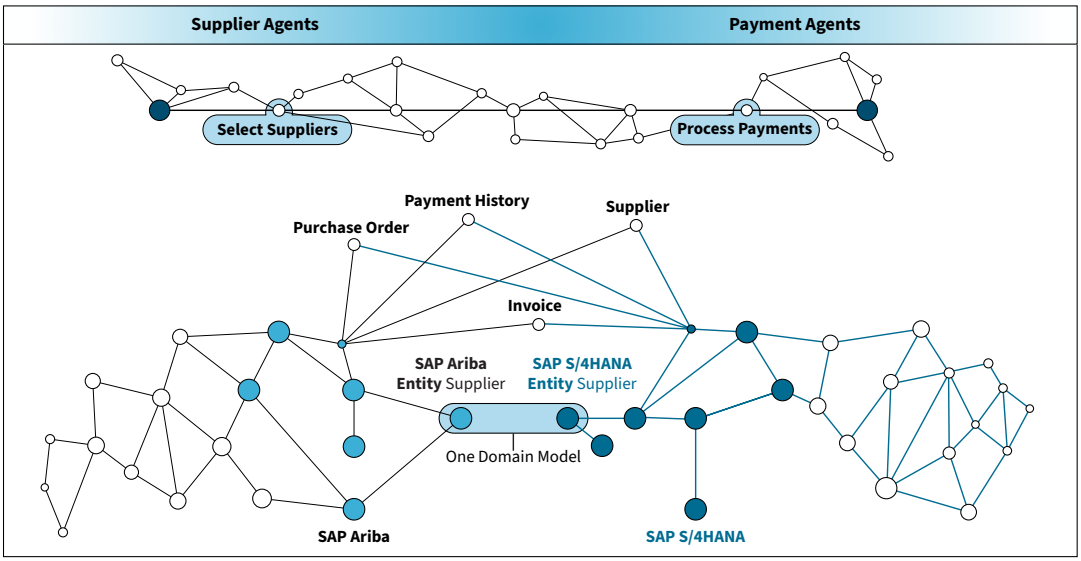


Figure 2.5: Evolution of SAP Knowledge Graph

SAP Knowledge Graph is the translator that connects the rigid, technical world of database tables to the flexible, human-like reasoning of an AI. It is the semantic layer that allows an agent to understand the *meaning* of business data, not just its structure or where it is stored.

To understand why SAP Knowledge Graph is the critical missing piece for agentic AI, let's look at how data access has evolved:

- **Era 1: The relational database management system**
  First we had tables. Back in the age of relational databases, everything lived in neat rows and columns. If you wanted to know how a customer was related to a specific product,

you had to ask a developer to write a monster SQL query with five different joins. It was precise, but painfully slow and completely unforgiving if you made a mistake. This was deterministic but brittle.

- **Era 2: The data lake**
  Next we built the data swamp. We created massive data lakes, thinking we could just dump all our data into one giant repository such as Amazon S3 or Azure Data Lake Storage. This gave us flexibility, but we lost all the context. The relationships between data points got lost in the noise, which is why AI models trained on this kind of data often struggled to make sense of anything.
- **Era 3: The vector database (vector RAG)**
  Moving along, we tried vector search. With the rise of generative AI, we started chopping up documents and turning them into numerical vectors. This was a huge leap forward for semantic search, letting us ask things like, "Find me documents about shipping delays." However, vectors are fuzzy. They can tell you two things are similar, but they can't tell you exactly how they're connected.
- **Era 4: SAP Knowledge Graph (graph RAG)**
  That brings us to SAP Knowledge Graph. This is where we are now. We store data as a network of things or entities (the nodes) and the relationships between them (the edges). For example: Customer A → [PLACED] → Order B → [CONTAINS] → Product C. The relationships aren't just implied; they're spelled out, creating a clear map for an AI to follow.

Why does connecting the dots (graph RAG) beat a simple search (vector RAG)? In the SAP context, vector RAG is insufficient for complex business questions. For the kinds of complex questions that actually run a business, a simple vector search just hits a wall. Imagine asking, "Which of our customers will be affected by the shipping delay of component X?" A vector search might pull up a few emails in which "component X" and "delay" are mentioned. However, it's almost guaranteed to miss the customer who hasn't complained yet, but whose entire production order depends on that one component.

How does SAP Knowledge Graph find the real answer? SAP Knowledge Graph can do what's called *multihop reasoning*, in which it literally traverses the connections to find the answer:

1. It starts with *component X* and sees it's part of *stock transfer order 100*.
2. It follows that link to see *order 100* is heading to *plant Hamburg*.
3. From there, it sees that *plant Hamburg* is using the component to make *finished good Y* for *sales order 999*.
4. It knows that *sales order 999* belongs to *customer Z*.
5. Therefore, it arrives at the final answer: "Customer Z is affected."

This ability to walk the graph allows an agent to provide deterministic, factual answers derived from deep, multilevel relationships that no simple text search could ever find.

Now let's look at some key aspects of SAP Knowledge Graph:

1. **The architecture: Connecting SAP Business Data Cloud**
   SAP Knowledge Graph works like a smart map for your data. It isn't another database you have to manage. Think of it as a smart, semantic map that sits right on top of your SAP Business Data Cloud.

   It has a semantic overlay and speaks plain English. The graph translates cryptic table names into simple business terms. Instead of forcing an AI agent to know that it needs to query table `KNA1` for customer data, the graph just provides an entity called *Customer*. This lets the LLM work with natural questions like "Show me our customers in Germany," which the graph then translates into the right technical query behind the scenes.

   The best part is that the graph doesn't force you to copy petabytes of data into yet another system. There is no data duplication! It leverages SAP Datasphere's federation capabilities to create a virtual map on top of your existing systems. The map itself is lightweight, but the data it points to is your entire transaction history, which stays right where it is in SAP S/4HANA or SAP BW/4HANA. This means you always get real-time answers without data lag.

2. **Grounding the agent (the antihallucination shield)**
   For an AI agent, SAP Knowledge Graph is the ultimate antihallucination shield, or grounding tool. It is consulted at two points in the loop, before the agent commits to a plan and after:

   - **Pregeneration check**
     This is more like a preflight reality check. Before an agent even tries to solve a problem, it can check the graph to understand the state of the world. Does this product actually exist? Is this plant open today?
   - **Postgeneration validation**
     Now the postflight validation: If the agent comes up with a plan like "Transfer stock from Plant A to Plant B," the system can check it against the graph. Is there even a transport route defined between A and B? If not, the graph flags the plan as impossible before the agent tries to execute it. Policy and safety screening is a separate concern, enforced by the trust layer's input and output filtering modules rather than by the graph. Chapter 9 covers it in detail.

3. **Future-proofing: The enterprise brain**
   As the graph grows, it evolves from being just a map of your data into a true enterprise brain that understands your actual business processes.

SAP Knowledge Graph is evolving to capture business events and their causal links. When an order is created, that event can trigger a credit check, which may in turn trigger a fulfillment hold or release. Modeling these event-to-event relationships lets an agent reason about consequences. This means, for example, that an Order-to-Cash Agent seeing a new high-value order can proactively confirm the customer's credit standing before the process stalls.

Simulating the future allows agents to run simulations and analyze the ripple effects of their decisions. They can finally answer questions like, "If I switch this supplier, what other processes down the line are going to break?"

2.3.4 SAP Business Data Cloud: The Unified Data Foundation

Think of SAP Business Data Cloud as the fuel for smart decisions. An AI agent is only as good as the data it runs on. If the generative AI hub is the engine and SAP Knowledge Graph is the GPS, then SAP Business Data Cloud provides the high-octane fuel needed to actually get somewhere using the actual, live business data that fuels every decision.

For decades, enterprise data architectures were designed for reporting, offering static, retrospective dashboards populated by nightly batch jobs in an extract, transform, and load (ETL) process. However, agentic AI demands a fundamental architectural shift. An agent tasked with optimizing inventory cannot work with yesterday's numbers. It requires data that is available in real time, federated, and semantically rich: data that is live, connected, and loaded with business context.

Figure 2.6 shows SAP Business Data Cloud's comprehensive strategy for enterprise data, highlighting a clear path to a data fabric across multiple compute options.

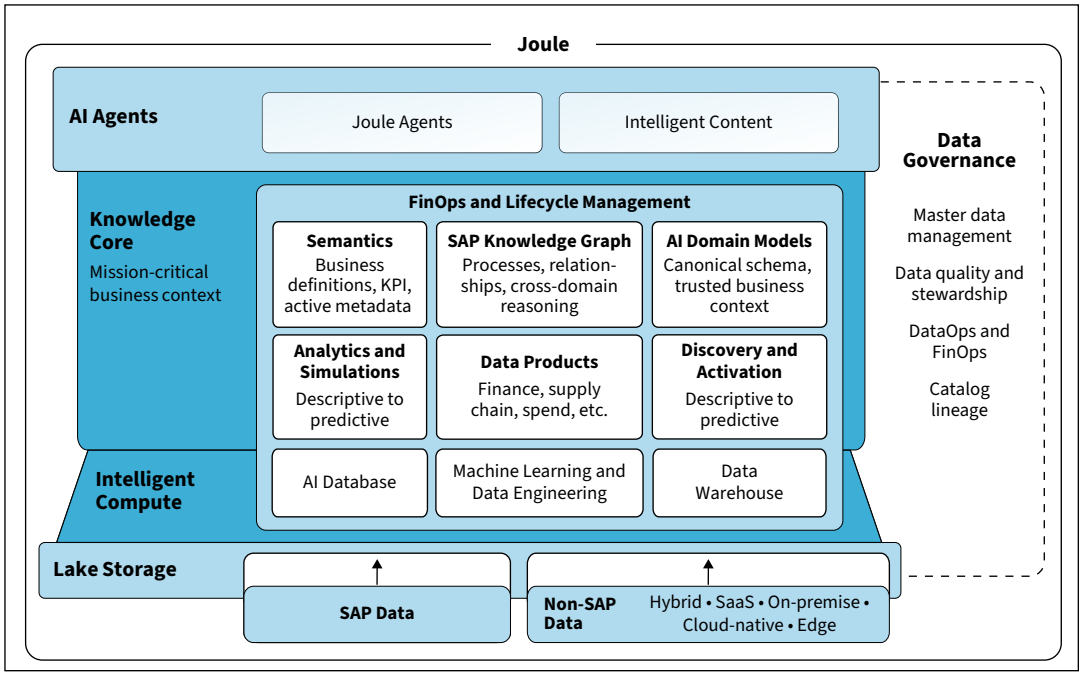


Figure 2.6: SAP Business Data Cloud Enterprise Data Strategy

SAP Business Data Cloud was built for this new world. It moves beyond the monolithic data warehouse paradigm to a business data fabric. This fabric acts as the fuel pump for the

agentic AI ecosystem, ensuring that agents act on the single source of truth rather than fragmented copies. Post-SAPPHIRE, SAP Business Data Cloud's foundation has been strengthened with deep Databricks integration and a lakehouse architecture, along with SAP's acquisitions of Reltio (master data management) and Dremio (lakehouse), broadening the range of SAP and non-SAP data the fabric can govern.

Let's dive into the different aspects of SAP Business Data Cloud:

- **The architectural shift: From data warehouse to data fabric**
  As outlined in the modern data architecture, traditional data warehouses just can't keep up with AI. They struggle with the volume and variety of data required for AI. They create what we call a *context gap*. The moment you rip data out of a system like SAP S/4HANA and dump it into a generic data lake, you strip away all the business logic. A column named `LIFNR` becomes just a meaningless string, losing its identity as a vendor.

  SAP Business Data Cloud, powered by SAP Datasphere, fixes this by creating a business data fabric. Instead of the old ETL model that required moving mountains of data around, SAP Business Data Cloud leaves data right where it lives—in SAP S/4HANA, SAP BW/4HANA, or even non-SAP systems—and creates a virtual layer to access it in place. The focus is on connecting the data, not copying it.

  The semantic layer is a magic translator that serves as the critical bridge for agents. It translates technical jargon (cryptic database schema) into plain business language. It defines, for example, that *Revenue = (Sales - Returns) * Currency Conversion*. When an agent asks for *revenue*, therefore, it doesn't have to write a line of SQL; it just uses the preapproved definition, guaranteeing it calculates the number exactly the way your CFO would.

- **The currency of agency: Data products**
  In the agentic era, we stop thinking about tables and start thinking about data products. A *data product* is not just a dataset or a pile of raw data. It is a reusable, trusted, and self-contained unit of value.

  For anyone building an AI agent, this is a game changer, a massive productivity shift. Instead of trying to reverse engineer messy table joins, you simply have the agent subscribe to a certified data product.

  According to SAP's architectural guidelines, a data product consists of three layers:

  - *The data (the core):* The actual information (e.g., open sales orders), federated or cached for performance.
  - *The metadata (the context):* Technical and business metadata that tells the agent what this data represents, its freshness, and its lineage—or what the data means, how fresh it is, and where it came from.
  - *The governance (the contract):* Embedded policies regarding who can access the data product (e.g., only HR agents can read salary data). This is the built-in governance about who can use it.

SAP Business Data Cloud even comes with a marketplace for these data products. If you're building a Supplier Risk Agent, you can just browse the marketplace, find the Supplier 360 data product, and plug it in. This promotes reusability; for example, the same data product used by the human CFO for reporting is used by the AI agent for reasoning, thus guaranteeing that everyone is on the same page.

- **Modernization: The SAP BW bridge**
  We know that many organizations have spent decades building valuable logic in SAP BW and SAP BW/4HANA, and rebuilding it from scratch would be a nightmare. SAP Business Data Cloud tackles this head-on with the SAP BW bridge, letting you modernize without throwing away your investment, thus enabling a modernization in place strategy.

  An agent can answer complex questions by combining live operational data from SAP S/4HANA with historical analytical data from the SAP BW bridge. Two different actions are involved here:

  - User query—for example, "Why are our Q3 margins lower than the five-year average?"
  - Agent action—for example, the agent fetches real-time Q3 data via federation (SAP S/4HANA) and fetches the five-year trend from the SAP BW bridge, synthesizing both into a single answer.

  You do not need to rewrite your SAP BW queries. The bridge exposes them as data products within SAP Datasphere, making legacy logic instantly accessible to modern AI agents.

- **Governance as metadata that's built in, not bolted on**
  In an agentic system, governance cannot be a manual gatekeeper; it must be computational. As detailed in SAP's "Classifying Metadata" learning journey (see *http://s-prs.co/v633704*), metadata is the backbone of automated governance. In a world with autonomous AI agents, governance can't be handled via a manual approval form. It must be automated and intelligent.

  SAP Business Data Cloud uses its metadata catalog as the backbone for this. It uses the SAP Datasphere catalog to crawl and index data assets. It tags data with sensitivity labels (e.g., confidential or PII). When an agent attempts to access a data product, the underlying governance policy is checked automatically. If an agent with a logistics persona tries to access HR payroll data, the SAP Business Data Cloud layer denies the request at the query level, preventing unauthorized leakage regardless of the prompt used.

- **Fueling the generative AI hub: The vector engine**
  Finally, SAP Business Data Cloud is the direct integration point for the generative AI hub. Agents deal with more than just numbers; they read contracts, PDFs, and emails. SAP HANA Cloud includes a native vector engine that stores *embeddings* (vector representations) of unstructured data alongside structured business data. Hybrid search enables the holy grail of enterprise search. An agent can execute a single query that filters by structured data (e.g., "Vendors in Germany") and searches by semantic similarity (e.g.,

"Contracts with high liability clauses"). This combination of structured and unstructured data is what allows SAP agents to perform complex due diligence tasks that generic AI cannot touch.

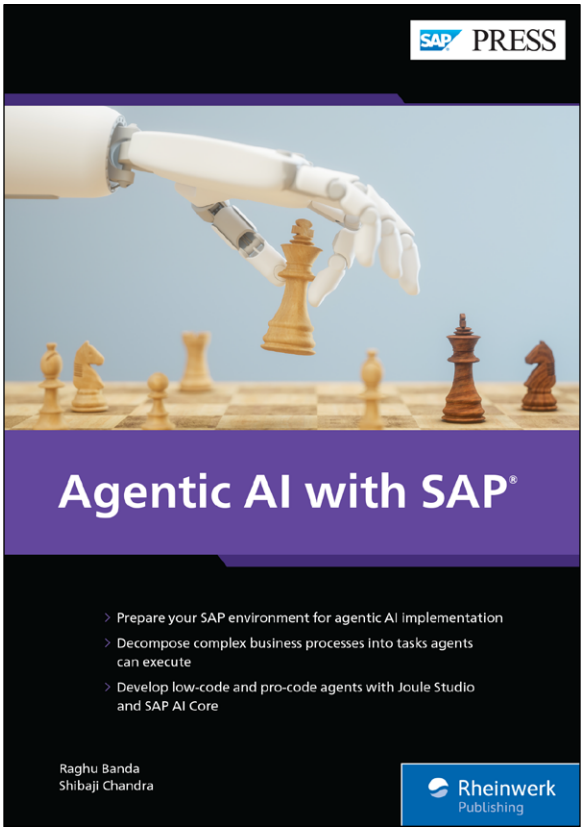
2.4 Summary

In this chapter, we peeled back the marketing layers of generative AI to reveal the engineering reality of agentic AI. We established that an agent is not merely a chatbot that summarizes text; it is a probabilistic orchestrator capable of managing the complexity and ambiguity of modern enterprise work. We also explained that possessing a powerful AI model does not tell you where to place it in, say, a global order-to-cash process. Having access to the generative AI hub does not explain how to decompose a monolith into agent-sized microservices. A pile of bricks is not a house; a stack of AI services is not a landscape.

We laid out the core principles of the Autonomous Enterprise. We established that agentic AI isn't just another upgrade to automation; it's a complete shift in control. We moved beyond the rigid if-then logic of traditional workflows to the goal-seeking behavior of intelligent agents, all driven by the constant five-stage cycle of perception, reasoning, planning, action, and reflection. We also broke down the agentic stack that powers it all: The generative AI hub provides model-agnostic reasoning, SAP Domain Models (SAP-RPT-1.5 and SAP-ABAP-1) supply business intuition, SAP Knowledge Graph keeps it grounded across 7.3 million data fields, and SAP Business Data Cloud feeds the system with real-time, trusted information. We discussed how tool-calling is standardized through MCP, how agents collaborate through A2A, how Joule Studio 2.0 and its runtime provide the build-and-run environment, and how SAP AI Agent Hub governs the entire fleet. Most importantly, we saw how all this power is secured by the AI trust layer within the govern pillar of SAP Business AI Platform, ensuring that autonomy and reliability go hand in hand.

However, understanding the parts of a single agent is just the first step. The real challenge is orchestrating them all into a smart, scalable system. In Chapter 3, we'll shift our focus from the technology itself to the enterprise architecture that holds it all together. After all, possessing the engine is not the same as winning the race. The experienced enterprise architect has navigated tides of disruption before, from the dot-com boom to the postpandemic supply chain crisis, yet the age of agency brings a disruption that is permanent and profound. In the next chapter, we will answer a critical question: Is traditional enterprise architecture too slow for the speed of AI? We will explore how the enterprise architect role evolves from a governance gatekeeper to a strategic orchestrator, and we will define the *augmented enterprise architect*, a professional who not only designs AI systems but is empowered by them, using real-time, AI-powered tooling to navigate complexity at a scale previously thought impossible.

The engine is built. Now it is time to design the blueprint.



Raghu Banda, Shibaji Chandra

Agentic AI with SAP

- Understand how agents work in the SAP landscape and prepare your environment for agentic AI implementation
- Learn to decompose complex business processes into tasks agents can execute
- Develop low-code and pro-code agents with Joule Studio and SAP AI Core

www.sap-press.com/6337

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.

Author bios

Raghu Banda is an enterprise AI strategist and technology leader at SAP with more than 25 years of SAP experience and nearly three decades across enterprise technology, architecture, product innovation, and digital transformation.Shibaji Chandra is a visionary AI leader and principal enterprise architect at SAP with more than 23 years of experience driving enterprise-scale AI transformation.

ISBN 978-1-4932-2875-1 • 388 pages • 10/2026

E-book: $79.99

Rheinwerk Publishing