When AI Agents Go Shopping

A plain-English map of agent commerce, and the operating layer the protocol stack leaves unresolved

Kachyng Inc.

Part 3 of 8 · Patents & Patents Pending · July 2026

Executive summary

AI agents are beginning to transact: searching, carting, checking out, and paying on behalf of people and businesses. In under two years, more than a dozen protocols have appeared to standardize this, from MCP and A2A to UCP, ACP, AP2, TAP, Agent Pay, and x402. This paper maps all of them in plain English, in the order you would build them, through one running story: a customer, a sock shop, and its wool supplier.

Three findings emerge from the map. First, retail agent commerce is being standardized quickly. Discovery, checkout, customer approval, and payment verification each now have credible, well-backed protocols, although rivals overlap and several have already changed shape after first contact with the market. Second, every protocol on the map assumes the business's internal logic is already agent-ready; none of them defines it. Third, enterprise commerce (contracts, purchase orders, approvals, invoices, payment terms) remains almost entirely unaddressed, and that is where the money is: global B2B payment volume runs at roughly $89 trillion a year against about $6 trillion for retail e-commerce.

What is missing is an operating layer: the machinery that turns software built for humans into governed surfaces agents can safely operate, with identity, policy, approvals, limits, and evidence built in. The closing addendum describes how Kachyng's AGX and IDX address exactly that layer. The rest of the paper is the map that shows why it is needed.

Contents

Introduction

Over the past year, dozens of new standards and protocols have appeared for AI agents. Most articles explain one protocol at a time, which makes it nearly impossible to see how the pieces fit together, or which pieces are missing.

This paper walks the map in the order you would build the stack, one layer standing on the one before it, while treating none of these protocols as inevitable. Each layer gets two treatments: what it does, and **where it may break**. Several of these standards will win, several will merge, and several will quietly die — and several have already changed shape after first contact with the market. The map is worth learning either way: the surviving stack will be assembled from these parts. To keep it concrete, everything happens in one small story:

Every metaphor, protocol, and acronym in this paper is defined in the glossary appendix at the back.

The Agent Tech Stack at a glance

Layer	Technology	Metaphor	What it does
1	REST, GraphQL	**The Shop**	The application agents act on
2	OpenAPI, JSON Schema	**The Menu**	How your app lists what it can do
3	OAuth, OIDC	**The Badge**	Proves who the human user is
4	RFC 8693 token exchange	**The Handover Note**	Proves which user an agent acts for
5	MCP	**The Toolbox**	Hands your functions to AI as tools
6	A2A	**The Hired Hand**	Exposes a whole worker, not just tools
7	AG-UI, A2UI	**The Running Commentary**	Shows the user what the agent is doing
Support	schema.org, llms.txt, ANP	**The Shop Window**	How agents discover you exist
8A	UCP	**The Retail Vocabulary**	A shared language for shopping actions
8B	ACP	**The Marketplace Model**	Runs the store inside the AI platform
9	AP2	**The Permission Slip**	Captures the customer's approval to pay
10	TAP, Agent Pay	**The Gatekeepers**	Card networks verify the agent is allowed
11	x402	**The Meter**	Lets software pay software per request
12	the unsolved layer	**The Wholesale Aisle**	The same, but between businesses
Spans all	Kachyng AGX	**The operating layer**	Governs every layer — identity, limits, approvals, evidence

- **Sock Shop: a small online store** selling socks.
- **Alice: a customer with** an AI assistant.
- **Wool Supply Co: Sock Shop's wholesale** supplier.

Throughout the paper we follow what happens when Alice tells her assistant:

> *"Buy me six pairs of wool running socks under $60."*

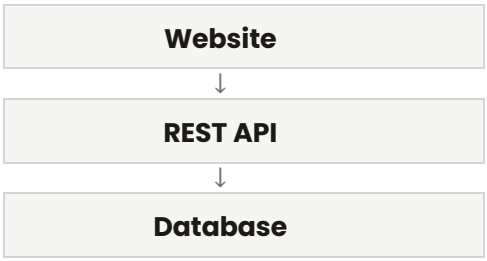
The short version, up front: most of the new protocols help agents discover a business, communicate with it, prove consent, and pay. None of them transforms the business's raw APIs and buried operating rules into a governed set of actions that an agent can safely understand, select, and execute. That missing transformation, governance, and execution layer is where Kachyng sits. The rest of this paper earns that sentence one layer at a time.

The starting point: AI doesn't replace your application

Before any protocol, remember one thing, because it defuses most of the anxiety in this subject:

AI doesn't replace your application. It sits on top of it.

A typical online store already looks like this:



Every protocol in this paper is simply a new way for outsiders, human or machine, to interact with that same application. Nothing below gets thrown away.

One sentence to hold onto for the whole climb: **applications expose functions; businesses operate through outcomes.** The distance between those two is where most of this paper happens.

Layer 1 — The Shop: your application (REST, GraphQL)

Sock Shop is, first, just software with functions: view products, check inventory, create an order, cancel an order, process a refund. These are exposed through an API: an Application Programming Interface, meaning the set of buttons other software is allowed to press (usually REST or GraphQL).

> *Example:*
>
> GET /products
>
> POST /orders
>
> POST /returns

Nothing about AI changes this layer. It remains the system of record. Everything an AI agent will ever do to Sock Shop presses these same buttons.

Layer 2 — The Menu: your application describing itself (OpenAPI, JSON Schema)

Humans can read documentation. Software can't. So the API needs a machine-readable description of itself: what endpoints exist, which fields are required, what comes back, what the validation rules are.

That is what **OpenAPI**[1] does, with its companion **JSON Schema** (a standard way of describing data shapes). Think of it as an instruction manual written for software.

This file looks mundane, but much of the map above rests on it: toolboxes are generated from the menu, integrations are validated against the menu, and agent behavior is shaped by what the menu declares. A sloppy menu produces sloppy integrations, and without one, every connection becomes more custom.

> *Example: Sock Shop's menu says "createOrder takes a list of items and a shipping address, and returns an order number." An AI reading that one line now knows how to place an order without a human explaining it.*

[1] OpenAPI Specification, maintained by the OpenAPI Initiative under the Linux Foundation since 2015, evolved from Swagger; JSON Schema, a community standard developed through IETF drafts for describing data shapes.

Where it may break: OpenAPI describes ***syntax*, not *meaning***. It can say a field is a string; it cannot say the field is confidential, or that an action requires approval, or that a price depends on who is asking. Any tool or interface generated from the specification inherits that blindness unless business meaning, policy, and authority are supplied separately. OpenAPI tells software how to call a function; it does not decide whether that function should be called.

Layer 3 — The Badge: who is logged in (OAuth, OIDC)

Alice signs into Sock Shop. How does the site know who she is, and what she may do?

The established answers are **OAuth**[2] and **OIDC** (OpenID Connect), the standards behind every "Sign in with Google" button. Together they hand Alice a badge that says who she is and what doors it opens.

> *With her badge, Alice can see her own orders, update her address, view her loyalty points. She cannot see another customer's orders, issue refunds, or reach employee systems.*

Identity existed long before AI. AI builds on top of it.

Where it may break: OAuth was designed for a human in a browser clicking "allow." Its consent model and its coarse scopes have no native concept of an autonomous agent making thousands of decisions per hour. It survives into the agent era as plumbing: necessary, proven, and nowhere near sufficient.

Layer 4 — The Handover Note: who an agent works for (RFC 8693)

Now something genuinely new happens. Alice doesn't buy the socks herself; her assistant does. The badge alone is no longer enough, because Sock Shop needs proof of a *chain*: this is an agent, it acts on behalf of Alice, and here is the paperwork.

[2] OAuth 2.0, IETF RFC 6749, 2012; OpenID Connect, OpenID Foundation, 2014. The web's standard authorization and login layers.

The standard for this is **RFC 8693 token exchange**[3]. Think of it as a signed handover note stapled to Alice's badge. Instead of the assistant claiming *"I am Alice,"* it proves *"I am Alice's assistant, and Alice authorized me."*

Almost every explainer skips this layer. This paper will not, because it is the security spine of everything below. Without the handover note, Sock Shop cannot tell a helpful assistant from a thief holding a stolen password. The handover note is also not the same thing as Layer 9's permission slip: the note proves who the agent works for; the permission slip proves what the customer approved.

(Keep this idea separate from customer *consent*, which arrives at Layer 9. The handover note proves *who the agent works for*. It does not yet prove *what the human agreed to spend*.)

Where it may break: technically sound, practically orphaned. Identity-provider support for token exchange is patchy and inconsistent, there is no ecosystem convention for how a receiving business should read a delegation chain, and the standards for preserving full multi-hop delegation chains across domains remain incomplete[4], so every implementation is bespoke. The right idea, still waiting for an adoption engine.

Layer 5 — The Toolbox: exposing your functions to AI (MCP)

Suppose an outside AI wants to check inventory. Without a standard, it has to learn Sock Shop's particular API, and every company's API is different.

MCP (**Model Context Protocol**, published by Anthropic)[5] fixes this by exposing your existing functions as *tools* an AI can pick up and use. Instead of calling `GET /inventory?id=123`, the AI simply calls `CheckInventory()`.

MCP is an adapter. Your application stays exactly the same; only the interface changes. The visiting AI does all the thinking; Sock Shop just performs the requested action.

[3] RFC 8693, "OAuth 2.0 Token Exchange," IETF, January 2020 — the standard mechanism for exchanging one token for another that carries delegation context.
[4] "OAuth Identity and Authorization Chaining Across Domains" (draft-ietf-oauth-identity-chaining), IETF OAuth Working Group, Internet-Draft, still active as of mid-2026 — combines RFC 8693 token exchange with RFC 7523 JWT authorization grants to preserve identity across trust domains; not yet an RFC.
[5] Model Context Protocol specification, modelcontextprotocol.io — introduced by Anthropic in November 2024; now developed under open, multi-vendor governance.

> *Example: Alice's assistant, working through a return, picks up Sock Shop's `StartReturn` tool and uses it: a visiting system operating your functions, one at a time, under your terms.*

Where it may break: MCP has the strongest real adoption of any agent door, and now provides transport authorization and stable enterprise-managed authorization[6], but it still does not define the business authority *inside* a tool call: which fields may be returned, which limits apply, which policies must be evaluated, and when a human approval gate must fire. It also strains at enterprise scale. A large business exposes hundreds of functions across home-built systems, customized packages, and SaaS (Software as a Service, rented applications) products, each with different ownership, permissions, and release schedules. Decades of accumulated systems, the IT sprawl every CIO knows, do not become tidy just because an agent is calling them. Salesforce's own MuleSoft research puts the average enterprise at 897 applications with only 29% of them integrated, a statistic Marc Benioff leans on when arguing that fragmented stacks cannot simply be wired to AI one system at a time.[7] An agent presented with too many tools selects poorly, and the company inherits a federation of surfaces that must be versioned, secured, and monitored. Tool sprawl looks like a performance problem. It is a governance problem: which functions should *this* agent see, for *this task and this tenant, and who decided?* MCP exposes. It does not select.

Layer 6 — The Hired Hand: exposing a worker instead of tools (A2A)

Sometimes you don't want an outside AI deciding every step inside your business. Instead of a toolbox, you expose an intelligent *worker*.

That is what **A2A** (**Agent-to-Agent**, created by Google and since donated to the Linux Foundation)[8] provides. The caller doesn't say "check inventory, create refund, notify shipping." The caller says one thing, *"handle this return"*, and Sock Shop's own agent decides the steps: verify the purchase, check the return policy, approve the refund, notify the warehouse, send confirmation. The caller delegates the job rather than prescribing every tool call; the remote agent controls the internal execution and decides how much progress and evidence to expose.

---

[6] MCP authorization specification, revisions 2025-06-18 and 2025-11-25 — the June revision defines OAuth 2.1 transport authorization (MCP servers as OAuth resource servers, RFC 9728 discovery, RFC 8707 resource indicators); the November revision adds Client ID Metadata Documents and the Enterprise-Managed Authorization (Cross App Access) extension.

[7] MuleSoft Connectivity Benchmark Report, 2025 edition (Salesforce): the average enterprise manages 897 applications, of which 29% are integrated. The 2026 edition updates these to 957 applications, 27% integrated.

[8] A2A was launched by Google Cloud in April 2025; the Linux Foundation announced the A2A project on June 23, 2025. Its Technical Steering Committee includes AWS, Cisco, Google, IBM Research, Microsoft, Salesforce, SAP, and ServiceNow (a2a-protocol.org).

> *Toolbox: the visitor operates your tools, step by step. Hired hand: the visitor hands over the whole job, and your system returns the result.*

Where it may break: a chicken-and-egg problem. A2A asks every business to build and operate a *competent agent* before any caller demand exists. And the delegation itself introduces a trust problem: the caller does not control the remote agent's internal reasoning or tool sequence, and therefore needs evidence, policy, and auditability around the result. Its move to open governance under the Linux Foundation reduces vendor-control risk, but it does not solve the harder economic problem: who builds, secures, supervises, and pays for the business-side agent?

Layer 7 — The Running Commentary: showing progress to users (AG-UI, A2UI)

Alice starts a return on Sock Shop's own website. The AI may take twenty seconds. During that time the page should say something:

> *Looking up your order… Checking return policy… Contacting warehouse…* ***Refund approved.***

Rather than every company inventing its own live-update format (which is exactly what everyone did in 2025), AG-UI (**Agent-User Interaction protocol**)[9] defines a standard stream of events from an AI backend to a web page, including the moment the agent pauses and asks the human for approval. A related Google project, **A2UI**, standardizes declarative interfaces *generated by* agents; AG-UI can carry and render those interfaces: adjacent specifications, not duplicates.

Where it may break: a thin layer in a still-unsettled space: adjacent specifications rather than one finished standard, and the kind of pattern application frameworks tend to simply absorb. Useful to know; risky to bet on.

Supporting layer — The Shop Window: being found at all (schema.org, llms.txt, ANP)

Before any shopping can happen, Alice's assistant has to discover that Sock Shop exists. The technologies here are unglamorous and effective: search engines, structured **product feeds**,

[9] AG-UI (Agent-User Interaction protocol), an open protocol from CopilotKit for streaming agent activity, state, and interface events into user-facing applications; A2UI, Google's declarative generative-UI specification, announced December 2025, Apache 2.0, a2ui.org.

schema.org product markup, Merchant Center data (the same plumbing search engines have read for years), plus **llms.txt**[10], an emerging file that tells AI systems how to read your site.

One early addition to watch: **ANP**[11] (**Agent Network Protocol**), an attempt at a phone book so agents can find other agents and verify who owns them. Know the name; build nothing yet.

Where it may break: feeds work for retail discovery and are structurally broken for business-to-business. A feed treats commerce like a supermarket shelf: one product, one price, one stock number. In B2B there is no such thing as *the* price; there is *your* price, under *your* contract, at *your* volume tier, this quarter. And whether you may buy at all is not a stock question but an entitlement question. A feed is a snapshot of a negotiated reality. Even when frequently refreshed, it cannot independently determine which contract, entitlement, price, quantity rule, or approval applies to the buyer making the request. Uploading a catalog file and calling it B2B automation produces exactly one thing: a flood of misinformed agents placing orders the business has to reject. Publish rules, not feeds.

The Retail Assumption

One assumption quietly runs through nearly every commerce protocol on this map: *commerce begins with discovering products.* That is true when you are buying socks. It is rarely true when you are running an enterprise.

Enterprise procurement almost never begins with a search box. It begins with an approved supplier, a negotiated contract, an **ERP** (**Enterprise Resource Planning, the system of record** for a company's money and materials) event, inventory falling below a threshold, a maintenance schedule, an existing part number, a requisition. **The central business object is not a shopping cart. It is a purchase order, governed by contracts**, approvals, payment terms, cost centers, tax rules, compliance policies, and negotiated pricing. Those are not extensions to shopping. They *are* the commerce.

[10] llms.txt, proposed by Jeremy Howard (Answer.AI), September 3, 2024, llmstxt.org — a community proposal rather than a formal standard; adoption is concentrated in developer documentation, and Google has stated its search products do not read the file.
[11] ANP (Agent Network Protocol), an early-stage open-source community project for agent discovery and interconnection, agent-network-protocol.com.

Put the two mental models side by side:

Consumer commerce	Enterprise commerce
Search → Catalog → Cart → Checkout → Payment	Business need → Approved supplier → Contract → Pricing agreement → Requisition → Approval → Purchase order → Invoice → Settlement
Begins with discovery	Begins with a purchase request raised under a contract
Central object: the cart	Central object: the purchase order
A human decides at checkout	Policy decides, long before checkout
Human present at the transaction	Human present in the policy; exceptions escalate

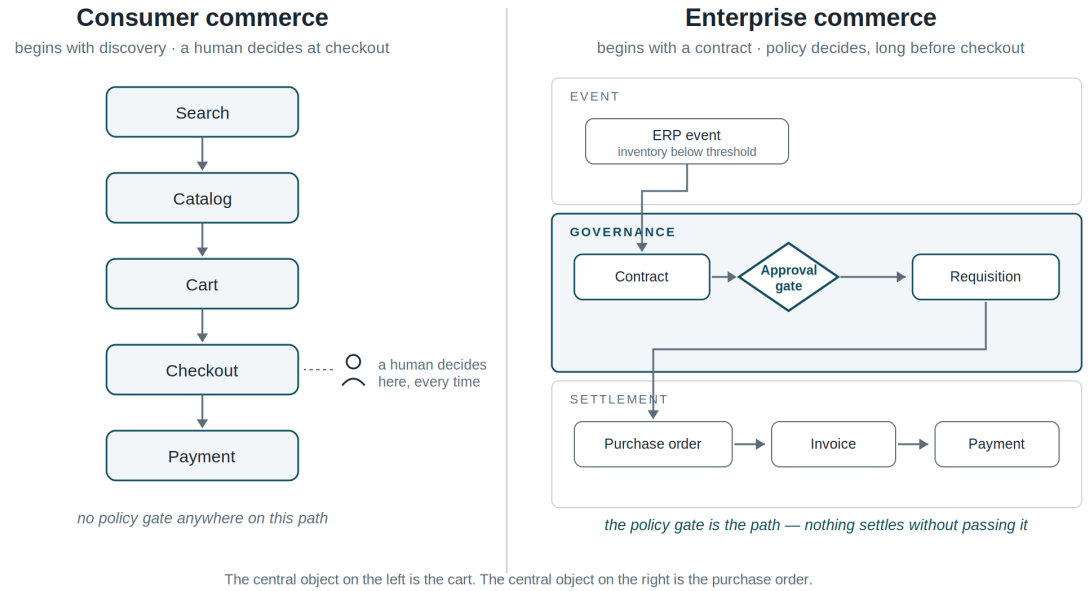


Figure 1. Two models of commerce. The consumer path is linear and ends with a human at checkout; the enterprise path routes everything through a policy gate before anything settles.

The two commerce protocols you are about to meet (**UCP** and **ACP**, in the next two layers) are currently expressed primarily through one flow: a customer or user agent interacting with a merchant through catalog, cart, checkout, and order objects. The flow an enterprise actually runs

looks nothing like it: an ERP event triggers a procurement agent, which negotiates with a supplier agent, which writes back to an ERP. Search may occur. Catalogs may exist. But neither is the governing object; the contract, the policy, and the purchase order are. A human may not be present at the moment of purchase, because the controlling judgment was encoded earlier in policy; only exceptions return to a person.

Read the next two layers, and then the whole map, with that distinction in mind, and the uncomfortable realization lands: half this protocol stack assumes shopping. The half of the economy that runs on purchase orders is, for now, largely unmapped.

Human in the loop, or autonomous?

The same split shows up on a second axis: *who is present at the moment of decision.*

The consumer protocols assume a human is nearby when it matters. Alice approves the purchase herself; the website pauses and waits for her; the spend is capped at what she personally agreed to (the signed permission you will meet at Layer 9). Autonomy, in that world, is bounded by a person standing next to it.

Enterprise operation cannot work that way. The reorder fires at 2 a.m. The procurement agent negotiates while the buyer sleeps. But autonomous does not mean unsupervised; it means the human's judgment has to move *earlier*: out of the individual transaction and into the rules. A named person approves the policy (spend limits, permitted suppliers, what data each agent may see) and the system runs inside it. Routine actions proceed without a human. Unusual ones stop and wait for one.

CONSUMER	ENTERPRISE
Human	Policy
Search	Business event
Cart	Agent
Checkout	Purchase order
Payment	Supplier

The human does not leave the loop. The human moves *up* the loop: from approving every transaction to approving the rules, and back down into the transaction only when a gate fires. Which raises the question this paper keeps circling: where are those rules and gates written down,

who approved them, and can you prove it afterward? Hold that question; it is the subject of the closing section.

Layer 8A — The Retail Vocabulary: UCP

Now the shopping trip itself. Before Alice's assistant can buy anything from Sock Shop, it needs to discover what the store supports. Can it browse products? Build a cart? Apply loyalty points? Process returns?

This is the purpose of **UCP** (**Universal Commerce Protocol), built by Google** and Shopify, backed by Target, Walmart, Visa, Mastercard, Amex, and Stripe[12]. It is the biggest piece on the map, and the one most often explained wrong: it is not a payment button. It is the **entire shopping conversation, standardized: discovery, cart**, checkout, order tracking. The protocol's ambition is that Sock Shop declares its commerce capabilities once, and any compatible agent can then shop there without a custom integration. And it runs over whichever door you already opened: the plain API, the toolbox, or the hired hand.

Where it may break: UCP has the strongest institutional backer list on the map, but a backer list proves institutional interest, not merchant adoption or architectural fitness. Its currently published primitives are unmistakably retail-shaped (catalog, search, cart, checkout, order) while complete support across industries, borders, and enterprise workflows remains roadmap work. UCP also acknowledges that discovery information is provisional and that eligibility and policy must ultimately be enforced by the merchant. In enterprise commerce, however, eligibility and policy are not checks performed at the end of checkout; they determine the transaction from its first line item. UCP also quietly turns merchants into interchangeable endpoints behind someone else's agent (the disintermediation fear that stalled comparison-shopping standards a generation ago), and the sponsor's commercial incentive is straightforward: the more merchants expose standardized commerce capabilities, the more useful Google Search and Gemini become as the places where commerce begins.

Layer 8B — The Marketplace Model: ACP

UCP has a rival architecture: **ACP** (**Agentic Commerce Protocol**), developed by OpenAI and Stripe[13], designed around commerce that begins *inside ChatGPT*. Merchants provide structured

[12] Universal Commerce Protocol specification, ucp.dev — announced January 11, 2026 at NRF; co-developed by Google and Shopify with Etsy, Wayfair, Target, and Walmart, and endorsed by 20+ partners including Visa, Mastercard, American Express, and Stripe. Published capabilities include catalog search and lookup, cart building, identity linking, checkout, and order management; an AP2 Mandates extension carries signed transaction evidence.
[13] Agentic Commerce Protocol specification, agenticcommerce.dev — co-developed by OpenAI and Stripe under Apache 2.0 (Stripe's documentation also credits Meta), announced September 29, 2025.

product information so ChatGPT can discover and present their products; ACP then connects the resulting customer interaction to the merchant's own commerce systems. The merchant still owns the order, payment processing, fulfillment, compliance, returns, and customer support. That makes ACP less a universal commerce layer than a **marketplace and distribution architecture**: ChatGPT is where the customer discovers and evaluates; the merchant fulfills the resulting demand.

Note how quickly this ground moves. OpenAI initially launched ACP around Instant Checkout inside ChatGPT; its current strategic emphasis is structured product discovery and merchant-owned checkout[14], although limited Instant Checkout experiences may continue during the transition. The protocol may remain; the commercial experience built on it is still changing, and it will not be the last protocol on this map redrawn after first contact with the market.

One historical naming trap is worth noting: IBM also used the acronym ACP, for its **Agent Communication Protocol**. That project has since been folded into A2A under the Linux Foundation and its original implementation archived[15], one more protocol reshaped after contact with the market. In current commerce discussions, ACP means the OpenAI–Stripe one.

Where it may break: ACP asks merchants to feed structured product data into an AI marketplace they do not control. That can be rational for simple retail goods, where discovery, price comparison, and checkout are relatively uniform. It is far less convincing for enterprise commerce, where there is no universal catalog, no universal price, and usually no shopping cart: the merchant gains access to ChatGPT's audience but cedes control of discovery and the customer interface. And note what both rivals share: each assumes commerce begins with a customer interacting with an AI assistant, precisely the assumption the Retail Assumption section questioned. In one line: ACP is a marketplace architecture; UCP is a retail-commerce vocabulary with ambitions of interoperability; neither is an enterprise procurement authority model.

Layer 9 — The Permission Slip: customer approval (AP2)

Alice said "buy socks." But did she really approve spending money? How much? At which merchant? Until when?

[14] OpenAI, “Buy it in ChatGPT: Instant Checkout and the Agentic Commerce Protocol,” September 29, 2025 — Instant Checkout launched with single-item purchases from U.S. Etsy sellers; discovery runs on structured product feeds under the OpenAI Product Feed Specification, with the merchant remaining seller of record for fulfillment, returns, and support.
[15] LF AI & Data (Linux Foundation), “ACP Joins Forces with A2A,” August 29, 2025 — IBM Research launched its Agent Communication Protocol in March 2025 for the BeeAI platform; the project merged into A2A, and BeeAI migrated to A2A.

AP2 (**Agent Payments Protocol**, from Google)[16] presents itself broadly, as a common language for accountable agent transactions across payment types, but one of its central mechanisms answers exactly this question: the **mandate**, a digitally signed authorization carrying verifiable proof of what the customer approved:

Field	Value
Maximum purchase	$60
Merchant	Sock Shop
Valid until	Friday

Sock Shop and the payment chain can verify the mandate before completing the purchase. The closest payments analogy is a signed authorization on file: before honoring the instruction, the merchant verifies the signature, not the messenger's word.

Notice this is different from Layer 4. The handover note proved *the agent works for Alice*. The permission slip proves *Alice agreed to this spend*. Two different questions; two different signatures. (These layers are conceptually separate even where implementations combine them; UCP, for example, documents AP2 integration for signed transaction evidence[17].)

Where it may break: the mandate asks the right question, but the user experience is unproven, and real-world behavior with consent prompts says people blanket-approve to make the friction go away, which quietly recreates the exact problem the mandate exists to solve. AP2 is strongest where a user can sign a bounded purchase mandate; the harder enterprise case (standing organizational authority delegated across budgets, roles, contracts, and multiple agents) remains much less developed.

[16] Google Cloud, "Announcing Agent Payments Protocol (AP2)," September 16, 2025 — developed with 60+ payments and technology organizations. Google donated AP2 to the FIDO Alliance on April 28, 2026, releasing v0.2 with support for "Human Not Present" autonomous payments.
[17] Universal Commerce Protocol specification, ucp.dev — the AP2 Mandates extension on the Checkout capability carries cryptographically signed transaction evidence.

Layer 10 — The Gatekeepers: payment network verification (TAP, Agent Pay)

Even with Alice's signed approval, one question remains at the till: how does Visa know the *agent itself* is legitimate?

The card networks are adding their own check at the door, before a card gets charged. Visa's version is **TAP** (**Trusted Agent Protocol**)[18]; Mastercard's is Agent Pay[19]. Both reach beyond a simple identity check: Agent Pay's current material, for example, extends into agent credentialing, programmatically enforced spending rules, and verifiable-intent evidence. Any honest map needs both.

The division of labor is not as clean as it sounds: AP2 focuses on verifiable intent and mandates, while TAP covers the cryptographic trust relationship between agent and merchant, including both the agent's identity and its associated authorization. The boundaries blur, and that blurring is itself part of the emerging fragmentation.

Where it may break: these are the most likely pieces on the map to "work," because the networks can simply mandate them, the way 3-D Secure was mandated. That is also the failure mode: mandated friction, uneven regional rollout, and an unreconciled overlap with Layer 9 that nobody has yet sorted out. Two verification layers asking adjacent questions is a recipe for double prompts and finger-pointing.

Layer 11 — The Meter: machine-to-machine payments (x402)

Not every purchase involves a human. Suppose another company wants access to Sock Shop's inventory data: not monthly, but one cent per request.

x402 (from Coinbase)[20] supports exactly this: software hits a service, receives the old HTTP status code *402 Payment Required*, pays a tiny amount automatically, and gets the data. This is software

[18] Visa, "Visa Introduces Trusted Agent Protocol," October 14, 2025 — developed with Cloudflare; agent-specific cryptographic signatures (HTTP Message Signatures / Web Bot Auth) let merchants recognize trusted agents from the browsing phase through payment.
[19] Mastercard Agent Pay, announced April 29, 2025 — Agentic Tokens built on the Mastercard Digital Enablement Service, with launch collaborators including Microsoft and IBM; Mastercard's companion Verifiable Intent standard, co-developed with Google, was donated to the FIDO Alliance in April 2026.
[20] Coinbase, "Introducing x402," May 2025, github.com/coinbase/x402 — revives the HTTP 402 Payment Required status code for automatic stablecoin payments; the x402 Foundation, announced with Cloudflare in September 2025, moved under the Linux Foundation in April 2026.

buying from software: nothing to do with Alice's socks, everything to do with a future where APIs are metered like utilities.

Where it may break: a clean design for a market that does not exist yet. Per-request settlement presumes an appetite (in practice, for stablecoin rails) that enterprise finance and compliance teams have not blessed. Worth watching; not worth building on this year.

Layer 12 — The Wholesale Aisle: business to business

Sock Shop's own inventory agent notices wool stock running low and wants to reorder from Wool Supply Co. The same pieces reappear at larger scale:

The **hired hand** (Layer 6) carries the goal between companies: *"quote me 500 units, delivery by the 20th."* UCP (Layer 8A) also stretches its retail vocabulary toward B2B: Wool Supply Co declares capabilities such as catalogs, quotes, and checkout, and Sock Shop's agent attempts to transact through them. But changing the buyer from a person to a company does not turn a cart-and-checkout model into enterprise procurement. And the **permission slip** (Layer 9) comes in a business flavor: signed proof that Sock-Shop-the-company authorized its agent to spend up to $5,000 a month at Wool Supply Co.

Now count what the protocol stack covers. The conversation between agents: covered (Layer 6). The catalog and quote vocabulary: partially covered (Layer 8A). The spending mandate: covered in outline (Layer 9). Everything else that makes the reorder safe (the contract lookup, the budget check, the approval chain, the three-way match, the payment terms) lives inside each company's systems and policies, and no protocol on this map describes, negotiates, or enforces any of it. That is not a criticism of the protocols; it is the boundary of what a wire format can do.

Walk one reorder through end to end and the gap becomes concrete. The trigger is not a search; it is a standing contract that says wool is bought from Wool Supply Co at negotiated prices, and a policy that says reorder when stock drops below two weeks of demand. The agent raises a purchase request, and someone (or some rule) must approve it against a budget. Approval produces a purchase order with terms attached: price breaks, delivery windows, Net 45 payment. Wool Supply Co answers with a sales order and, later, an invoice. Goods arrive and are received and inspected, the invoice is matched against the purchase order and the receiving record, and only then does payment go out, on terms, weeks after the "transaction" began.

An honest reality check: today, actual wholesale still runs on **EDI** (**Electronic Data Interchange**, the 1970s-era document standard) and email. This is the most aspirational corner of the map, which also makes it the most open.

Where it may break, and what fixes it: the feed fallacy from the Shop Window bites hardest here. The fix for B2B is not a smarter feed; it is flipping the model. Instead of publishing static data for anyone to consume, the business exposes a **governed decision interface**: the answer depends on who is asking, which contract applies, what the live systems say, and what authority has been granted, and every answer carries a recorded reason. Expose decisions. Enforce rules. That requirement points directly at the layer this paper closes on.

Why this matters

Consumer commerce is measured in millions of purchases. Enterprise commerce is measured in trillions of dollars[21] moving through contracts, procurement systems, and supply chains. If autonomous agents remain confined to retail shopping, they will improve convenience. If they become trusted participants in enterprise procurement, they will change how companies operate. That transition cannot happen without a way to govern autonomous authority.

The missing transition: from probabilistic reasoning to governed execution

Large language models are probabilistic reasoning systems. They interpret ambiguous instructions, weigh alternatives, and propose actions. Their outputs can also vary across repeated runs, including under settings intended to reduce variation[22][23].

That flexibility is useful. It is also why a model cannot be the final authority for an enterprise action.

A purchase order must ultimately be approved or denied. A refund must be issued, rejected, or escalated. A field must be returned or withheld. A transaction must proceed, stop, or wait for more information. Enterprise systems may contain forecasts, scores, optimization, and human judgment, but their control *points* must resolve into definite, enforceable outcomes.

The operating model therefore has a simple shape:

[21] Juniper Research estimates global B2B payment volume at roughly $89 trillion in 2024, reaching $124 trillion by 2028; global retail e-commerce, by comparison, runs on the order of $6 trillion per year.
[22] "Non-Determinism of 'Deterministic' LLM Settings," arXiv:2408.04667 — repeated-run accuracy instability of up to 15% under fixed settings where the theoretical maximum–minimum difference should be zero.
[23] "The Non-Determinism of Small LLMs: Evidence of Low Answer Consistency in Repetition Trials of Standard Multiple-Choice Benchmarks," arXiv:2509.09705 — consistent answers on only 50–80% of tested questions across ten repetitions at low inference temperatures.

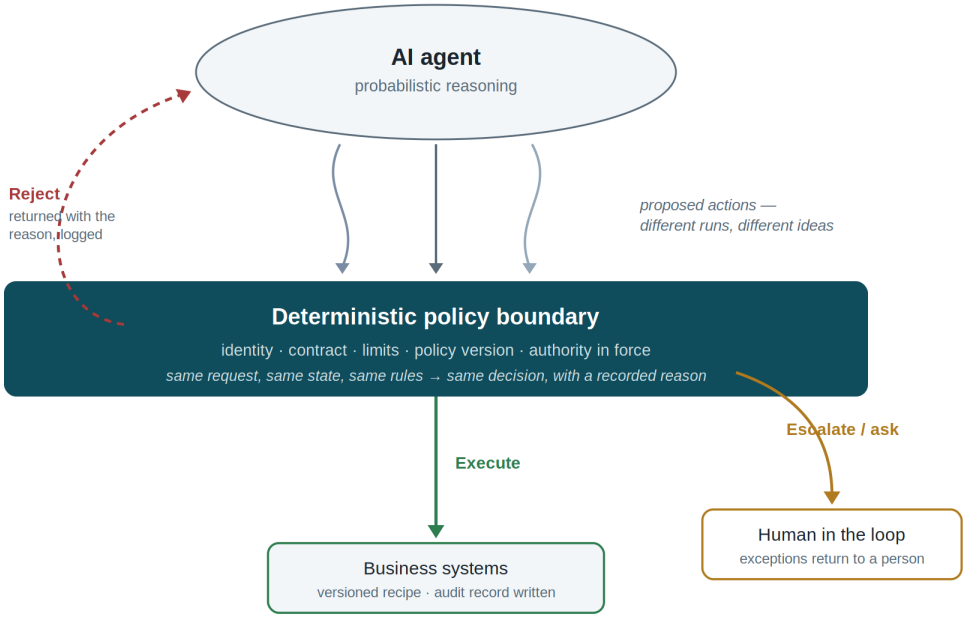


Figure 2. The deterministic policy boundary. The model proposes; the governed layer resolves every proposal into execute, escalate, or reject, with a recorded reason.

The model proposes. The governed layer decides what may actually happen. And for the same request, identity, business state, policy version, and authority context, that decision must be reproducible and explainable: which rule applied, who approved that rule, which data was considered, and why the action was executed, denied, or escalated.

This remains necessary even if model inference becomes perfectly reproducible. Engineering work can make the same model calculation produce the same output twice[24], but a repeatable recommendation is not automatically an authorized one. A refund does not become compliant because the model proposes it consistently. It becomes compliant because the business's policy permitted it under the facts and authority in force at that moment.

That is the distinction: **inference determinism is an engineering property. Decision determinism is a governance property.**

[24] He, H., Thinking Machines Lab, "Defeating Nondeterminism in LLM Inference," September 2025 — traces run-to-run variance to batch-dependent computation order and proposes batch-invariant processing.

Authority, then, is not the whole objective. It is one of the mechanisms (alongside contracts, entitlements, approval workflows, limits, and audit) that converts probabilistic reasoning into governed execution. The missing layer is that **deterministic execution boundary**: the place where an agent's proposed action becomes exactly one enforceable outcome (execute, reject, escalate, or request more information) before it reaches the business system.

Structured outputs, validated tool calls, and external verifiers all help, and mature agent stacks will use every one of them. But they constrain how the model speaks, not what the business permits. They complement the operating layer; they do not replace it.

What's missing

Walk back through the whole map and notice what was never established.

Who decided Sock Shop's agent may reorder $5,000 of wool, and where is that rule *written, versioned, and enforceable*? Which fields of Alice's record may the support agent actually see (her order history, certainly; her card number, never), and says who*, provably*? Can the support agent issue refunds, and up to what amount? Can it touch payroll? Negotiate contracts? Who approved those permissions, where are they recorded, and can an auditor verify, six months later, what rules the agent was operating under when it issued a refund at 2 a.m.?

Now line up the trust layers this paper covered, and the gap becomes precise:

- **The permission slip (AP2)** provides verifiable evidence of the **customer's** intent and authorization.
- **The gatekeepers (TAP, Agent Pay)** help merchants and networks recognize trusted **agents** and validate their associated authorization signals.
- **Nothing on the map** provides the **operator's** complete cross-system authority model: what an agent may see and do inside the business itself, under which versioned rules, with which internal approvals, provable afterward.

Today's protocols quietly assume those answers already exist somewhere. In most organizations they exist as scattered business rules, application logic, and manual approvals, an approach that stops working the moment agents become autonomous.

Every protocol in this paper makes agents better at *reaching* a business. None of them makes a business *safe to be reached*.

And notice the pattern running through every "where it may break" note above: each protocol *assumes* the business's internal rules already exist somewhere: machine-readable, current, enforceable. None of them creates that. Stated precisely: no broadly adopted protocol on this map

provides a complete enterprise operating plane for autonomous agents. The stack can only ever be as safe as the layer nobody built.

The unresolved plane is not merely an authority layer placed in front of existing APIs. It is an **operating layer** that transforms those APIs into governed business actions. It creates approved agent actions from raw functions, attaches the business meaning and restrictions that API descriptions cannot express, combines functions across systems into composite workflows, and presents each agent with a contextual menu of valid outcomes. It then executes the selected outcome through a versioned recipe, applying limits, approvals, exception handling, and audit evidence along the way.

Authority determines which actions appear and which recipes may run. Transformation makes those actions and recipes usable by agents in the first place. Authorization tooling does exist (identity platforms, policy engines, access-control frameworks), but it was built to govern *users inside a single application*, not to transform and govern autonomous action across a company's systems with versioned rules and a replayable record. The operating layer doesn't slot into the stack; it spans it.

> ***The principle: rules, not feeds.***
>
> ***The architecture: expose decisions; enforce rules.***

To be precise about what that means: not disclosing proprietary policy logic. It means exposing a governed interface whose answers (is this buyer eligible, what price applies, what quantity may be ordered, what approval is missing) are produced from live rules, identity, contracts, and authority rather than copied from a static catalog.

That layer is what Kachyng builds. And the arithmetic is straightforward: the more the rest of it succeeds, the more mandatory that plane becomes.

The complete stack

Discovery	product feeds · schema.org · llms.txt · ANP (early)
Your own site	AG-UI · A2UI
Commerce	UCP ⟷ ACP (OpenAI+Stripe) · rival retail models
Consent & trust	AP2 (customer mandates) · TAP (Visa) · Agent Pay (Mastercard)
Machine payments	x402
Agent doors	A2A (workers and agent tasks) · MCP (tools)

Identity	OAuth / OIDC · RFC 8693 delegation · the forgotten spine
Description	OpenAPI · JSON Schema
Core app	REST/GraphQL · database

> **AGENT OPERATING LAYER: spans and transforms the stack raw APIs → agent actions → contextual menus → composite recipes**
> governed by identity, policy, approvals, limits, evidence
> unresolved by every protocol above · Kachyng AGX

Final thoughts

AI commerce is not one technology. It is a stack, and each piece answers one question:

APIs expose business functions. OpenAPI describes them. OAuth identifies users. RFC 8693 lets an agent prove who it works for. MCP hands out tools; A2A hands over whole jobs. AG-UI narrates the work. UCP and ACP standardize shopping: a retail vocabulary and a marketplace architecture, respectively. AP2 carries verifiable evidence of the customer's authorization; TAP and Agent Pay establish which agents the networks trust; x402 lets software pay software.

One last observation: this map is not static. Several of these protocols have already been redrawn mid-flight (checkout ambitions retreating to discovery, corporate projects moving to foundations) because original intent met market reality and lost. Expect more of that; judge every protocol here by its trajectory, not its launch announcement.

The distinction worth remembering isn't the acronyms. It is the two mental models of commerce: retail (search, catalog, cart, checkout) and enterprise (contract, policy, approval, purchase order). Most of this stack was built for the first. An enormous share of the world's commerce moves through the second. And closing that gap takes more than a permission check placed in front of an API: raw functions must first become governed actions, or authority has nothing to govern.

Every protocol on this map assumes someone else solved enterprise authorization for agents.

No one has.

Permission precedes payment.

Addendum: Where Kachyng sits on this map

The Kachyng platform has four pillars; the two that matter for this paper are **AGX** and **IDX**.

AGX — the operating layer

AGX stands for **Agent Gateway Exchange**. It is the operating layer that transforms software written for humans into software operable by autonomous agents, turning raw APIs and buried operating rules into governed business actions. It is the plane the previous two sections argued was missing, built as a product. It does four jobs: it transforms software, models business outcomes, governs execution, and records evidence. Authority is its control model (determining which actions appear and which recipes may run), but it is one quarter of the product, not the whole of it.

The important thing AGX is *not* is a permission check bolted in front of an API. Enterprise software already contains thousands of functions and decades of business logic. The problem is that those functions were built for applications and developers, not autonomous agents: GetCustomer(), CheckCredit(), GetContractPrice(), `CreatePurchaseOrder()` say nothing about which functions belong together, in what order, with which inputs, under which contract, or when a human must approve. The rules that answer those questions are buried across ERP code, pricing engines, approval tables, configuration files, contract records, and employee knowledge, a surface no LLM can read, and no agent can safely be handed raw.

AGX transforms that surface. This is where the paper's opening sentence pays off: *applications expose functions; businesses operate through outcomes*, and AGX is the machinery that crosses the distance:

Agent actions. One generic function can become several approved actions for agents. A raw refund API becomes *refund damaged item*, *refund shipment failure*, *issue partial service credit*, or *escalate high-value refund, each with its own* permitted fields, limits, approvals, data visibility, and execution rules. Internally these may be implemented as variants of the same endpoint; to the agent, they are distinct business actions. The agent is not handed one powerful function and told to improvise; it is offered only the governed actions the business has defined.

Annotations. AGX lets the business capture, formalize, and attach the relevant rules to those actions as structured, governed context: field semantics, business rules, decision logic, visibility, human gates. The point is not documentation. A bare `POST /refund` cannot say why refunds exist, when they are allowed, which kind this one is, or who must approve it; annotation reconstructs the business semantics, the intent, that the function signature never carried, so the meaning no LLM can excavate from twenty years of application code arrives with the action itself.

Composites. An enterprise action is rarely one API call. *Reorder inventory* may cross six applications: check threshold, identify approved supplier, load contract, calculate contracted price, check budget, create requisition, route approval, issue purchase order, transmit, record evidence. AGX packages that sequence into one deterministic composite workflow. To the agent it is one available outcome; internally it is a fixed recipe spanning systems. And these are not wrappers: *approve customer (validate the customer, run compliance and credit checks, create the account, notify downstream systems, record evidence) exists nowhere* in the underlying software as a single function. A composite is a new business primitive that did not exist until the business defined it.

The contextual menu. What the agent finally sees is not four hundred tools. It is a rigid, short menu of valid outcomes *for this customer, contract, tenant, transaction, and current state, and the menu changes* when the context changes. This is more than convenience: AGX does not simply add rules, it removes ambiguity. Collapsing hundreds of technical functions into a handful of contextually valid business actions shrinks the decision surface presented to the model, and a model choosing among four governed actions behaves very differently from one improvising across four hundred tools. Constraint is not just safety; it is one of the reasons governed execution becomes possible at all.

The interface. The agent addresses that menu in plain language, not protocol syntax. The request arrives in whatever words come (reorder wool, *top up inventory*, *we're getting low*, *need another shipment), and resolution* happens in two steps: many utterances resolve to one **business intent**, and the intent resolves to the agent actions defined for it. Whether the systems underneath speak REST or GraphQL is an implementation detail the agent never sees. By design, no special vocabulary is required on the way in, and the agent does not improvise the execution path on the way out; the selected action runs through a versioned recipe with defined branches, stops, retries, and escalation rules. Natural language in; exactly one governed outcome out: execute, reject, escalate, or ask.

The whole transformation, at one glance:

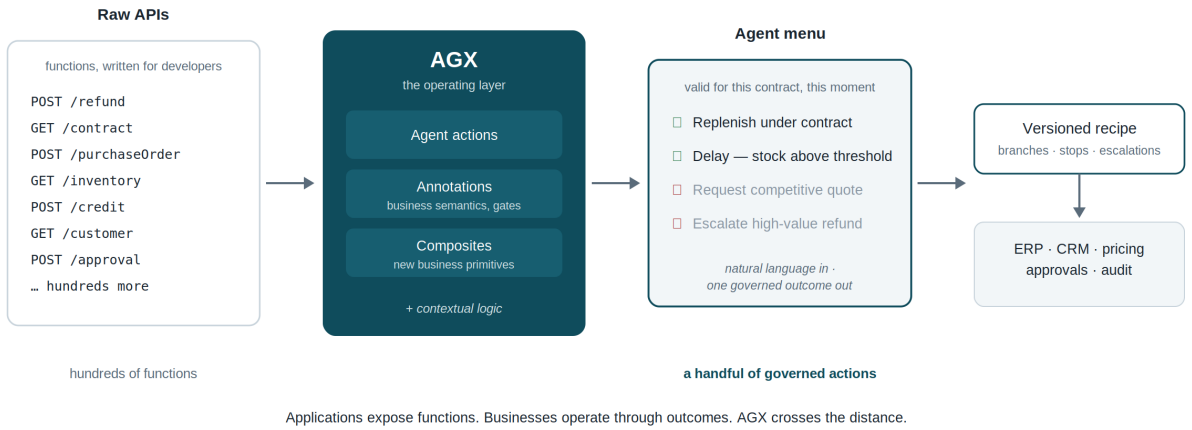


Figure 3. The AGX transformation pipeline. Hundreds of raw functions become a handful of governed, contextually valid business actions, executed through versioned recipes.

This is where the restaurant belongs. A diner at a multi-cuisine restaurant orders in plain speech (the Japanese set meal, the Indian vegetarian meal), and the house maps that request onto the menu, never onto the kitchen. The diner never enters the kitchen to pick knives, set temperatures, or waive the allergy rules. The menu constrains the diner's choice; the recipe constrains the kitchen's execution. AGX does both for agents: the agent remains probabilistic while interpreting the goal and selecting among valid options, and once it selects, the execution is governed by a predefined workflow rather than invented by the model.

> *In the story: Sock Shop's agent is told "replenish 500 units of wool." Without AGX it faces a dozen raw tools (search supplier, get price, create PO, send email, update ERP) and must invent the sequence, introducing variation with every run. With AGX it sees four actions: replenish under existing contract, request competitive quote, escalate due to contract expiration, delay because inventory remains above threshold. For this contract, this moment, only two are valid. It selects "replenish under existing contract," and AGX executes the recipe: validate supplier, load contract, apply price schedule, check quantity tier, check budget, create requisition, apply approval rule, generate the purchase order, transmit it, write the audit record. The agent chose the outcome. It never invented the implementation.*

The rules behind the menu are written down, versioned, and enforced. They cascade from the application down to the individual endpoint, agent, and tenant, and along the way they can only get

stricter, never looser. AGX creates a tamper-evident record tying each governed run to the policies in force, so the answer an auditor needs six months later exists the moment the action runs.

To be exact about the claim: AGX does not make the agent deterministic. It makes the business outcome governed, reproducible, and enforceable. AGX transforms software written for humans into software operable by autonomous agents by converting raw enterprise APIs and buried business rules into constrained, contextual, composite actions that agents can safely select and the enterprise can execute deterministically. The contrast with the map's most adopted door fits in one sentence: MCP makes functions callable by agents; AGX makes business outcomes operable by agents.

AGX is the operating layer. The deterministic business menu is how agents experience it.

Or in one sentence: every protocol on this map teaches AI how to reach software. AGX teaches software how to become operable by AI.

The reorder, revisited: Recall the Layer 12 walkthrough: contract, purchase request, approval, purchase order, invoice, receiving, payment on terms. With AGX in place, each gate becomes a governed action rather than a hope. The reorder request arrives in plain language and resolves against a contextual menu that contains only the actions this agent, under this contract, may take. The contract lookup and price check run inside the composite recipe, reading the live systems rather than a copied feed. The budget check fires as a deterministic policy (approve, reject, or escalate to a named human, with a recorded reason either way). The purchase order is created by the same recipe, the three-way match runs as a gate before payment is released, and every step leaves evidence an auditor can replay six months later. Same reorder, same systems; the difference is that no step depends on the model improvising.

IDX — the Badge Office

None of the above works without knowing who is at the table. **IDX** (**Identity Exchange**) is Layers 3 and 4 built as a product: it issues badges (OAuth/OIDC identity for humans *and* for agents) and stamps handover notes, building on RFC 8693 token exchange and extending it into a usable, verifiable chain of who acted for whom across the transaction, the part the bare standard leaves incomplete. The menu AGX presents depends on who is asking and on whose behalf; IDX is what makes that input provable rather than asserted. A related pillar, **KYA** (**Know Your Agent**), does for agents what banks' KYC (**Know Your Customer**) does for people: vetting the agent itself, the gatekeepers' job performed at the business's own front door.

How they fit together

IDX answers the first question every proposed action raises: *who is asking, and for whom, provably?* AGX answers everything after it: *which outcomes exist for this identity in this context, which one may run, under which recipe, with what record?* One identifies the diner; the other owns the menu and the kitchen. Together they are what turns "safe to be reached" from a slogan into infrastructure.

Fair objections

The integration objection: If the average enterprise runs 897 applications and integrates 29% of them, who is going to annotate and group those systems into governed actions? The same teams that already maintain API gateways and integration platforms, one workflow at a time. AGX does not require boiling the ocean; it requires picking the workflows agents will touch first (a reorder, a refund, an account approval) and governing those. The map is built incrementally, and each governed workflow pays for itself on its own.

The maintenance objection: What happens when the contract changes in the ERP? Nothing needs to be re-published, because AGX references rules; it does not copy them. The contextual menu is computed from live systems at request time, so a changed price break or a suspended supplier shows up in the very next menu. Feeds go stale; references do not. That is the practical meaning of rules, not feeds.

This paper is Part 2 of a four-part series. IDX is the subject of Paper 1, The Future of AI Agent Identity. AGX is covered in full technical depth in Paper 3, the AGX architecture whitepaper.

Appendix — Glossary

Three groups: the metaphors used on the map, the protocols and standards behind them, and Kachyng's own terms.

The metaphors

Metaphor	What it actually is
The Shop (Layer 1)	Your application: the software and database that do the real work. REST, GraphQL.
The Menu (Layer 2)	A machine-readable description of your API. OpenAPI, JSON Schema.
The Badge (Layer 3)	Proof of who is logged in. OAuth 2.0, OIDC.
The Handover Note (Layer 4)	Proof an agent acts for a specific user, with scoped authority. RFC 8693 token exchange.
The Toolbox (Layer 5)	Your functions exposed as tools an AI can call. MCP.
The Hired Hand (Layer 6)	A whole worker exposed instead of tools; task in, result out. A2A.
The Running Commentary (Layer 7)	Live progress streamed to the user while an agent works. AG-UI, A2UI.
The Shop Window (Supporting layer)	Being found at all: product feeds, schema.org, llms.txt, ANP.
The Retail Vocabulary (Layer 8A)	A shared language for shopping: discovery, cart, checkout, orders. UCP.
The Marketplace Model (Layer 8B)	Checkout embedded inside the AI surface where discovery happens. ACP.
The Permission Slip (Layer 9)	Verifiable proof of what the customer approved. AP2 mandates.
The Gatekeepers (Layer 10)	Payment networks verifying agents at the door. TAP (Visa), Agent Pay (Mastercard).
The Meter (Layer 11)	Machine-to-machine micropayments over HTTP. x402.
The Wholesale Aisle (Layer 12)	Business-to-business commerce: contracts, purchase orders, approvals.

The protocols and standards

Protocol	Origin and purpose
OpenAPI / JSON Schema	Open standards for describing REST APIs and data shapes. The basis of the Menu.
OAuth 2.0 / OIDC	The web's login and authorization standards (IETF, OpenID Foundation).
RFC 8693	OAuth 2.0 Token Exchange. IETF, January 2020. The delegation mechanism behind the Handover Note.
MCP	Model Context Protocol. Anthropic, November 2024. Exposes application functions as tools for AI models.
A2A	Agent2Agent. Google, April 2025; a Linux Foundation project since June 2025. Agent-to-agent communication.
AG-UI / A2UI	Open specifications (2025) for streaming agent activity and interfaces to end users.
llms.txt	Jeremy Howard, Answer.AI, September 2024. A curated site index for AI systems; a community proposal, not yet a standard.
ANP	Agent Network Protocol. An early-stage open-source effort toward agent discovery and interconnection.
UCP	Universal Commerce Protocol. Google and Shopify with major retailers, January 2026. The retail vocabulary.
ACP	Agentic Commerce Protocol. OpenAI and Stripe, September 2025. Powers Instant Checkout in ChatGPT.
AP2	Agent Payments Protocol. Google, September 2025; donated to the FIDO Alliance, April 2026. Signed customer mandates.
TAP	Trusted Agent Protocol. Visa with Cloudflare, October 2025. Merchant-side verification of shopping agents.
Agent Pay	Mastercard, April 2025. Agentic Tokens carrying per-agent payment authority.
x402	Coinbase, May 2025; under the Linux Foundation since April 2026. Machine-to-machine payments over HTTP 402.

Kachyng terms

Term	Meaning
AGX	Agent Gateway Exchange: the operating layer that turns software built for humans into governed, agent-operable surfaces. The subject of Paper 3.
IDX	The Badge Office: identity and delegation infrastructure for AI agents. The subject of Paper 1.
KYA	Know Your Agent: vetting the agent itself before it transacts, the way banks vet customers.

About Kachyng

Kachyng is building Agentic Commerce Infrastructure: the identity, orchestration, and settlement rails that let autonomous AI agents transact on behalf of humans and enterprises. Its patent-pending IDX + KYA identity layer gives every agent an independent cryptographic identity bound to a human controller — with scoped delegation, runtime behavioral trust, and global revocation in seconds. The architecture is protected by a patent family of 31+ assets with priority dates spanning 2010 to 2026.

Proof materials — production walkthroughs, architecture diagrams, filed patent claims, and customer and processor evidence — are available under NDA. Contact us to schedule a briefing.

Kachyng Inc. · 535 Mission Street, San Francisco

kachyng.com · info@kachyng.com