OAP

An Open Protocol for Application Discovery and Trust

A Manifesto for the Post-App Store Era

Kevin Brooks
Founder, iStrata | NetGate | Webcelerate | myNewscast | Xuru | ProveXa
February 2026
DRAFT v0.2 — For Discussion

I. The Extraction Pattern

In 1995, Craigslist began extracting $30 billion in classified advertising revenue from local newspapers. It didn't replace what that revenue funded—the reporters, the city council coverage, the accountability journalism that held local power in check. It simply took the money and left the civic infrastructure to collapse. Twenty years later, most Americans cannot name their city council representative, and local government operates in functional darkness.

In 2008, Apple launched the App Store and began extracting 30% of every software transaction on the most important computing platform in history. It didn't replace what the open web provided—permissionless innovation, direct creator-to-user distribution, zero gatekeepers. It simply inserted a tollbooth on a bridge it didn't build and collected rent from everyone who crossed.

Both follow the same pattern: a new technology enables a gatekeeper to capture value from an ecosystem without replacing the function that value served. Classified ads funded journalism; the open web funded innovation. The gatekeepers extracted the economics and left the purpose behind.

This paper argues that we are at a moment where both extractions can be reversed—not by regulation or antitrust action, but by a technological shift so fundamental that the gatekeepers' business models become structurally irrelevant. It proposes a concrete, working protocol to ensure the new ecosystem remains open. And it presents that protocol not as a theoretical exercise, but as a functioning system with reference implementations, a working registry, and three production applications already using it.

II. The Cost Collapse

Something unprecedented happened in 2024 and 2025. The cost of building production software fell by roughly an order of magnitude, almost overnight.

Tools like Claude Code, Cursor, and GitHub Copilot didn't just make existing developers faster. They made a new class of builder possible: the domain expert who has spent decades understanding a problem but never had the technical ability to build the solution. The insurance adjuster who knows exactly how claims processing should work. The nurse practitioner who has watched EMR software fail her patients for thirty years. The condo board treasurer struggling with accounting software designed for CPAs.

This is not theoretical. In a five-month period between late 2025 and early 2026, a single founder with no engineering team built three production applications using Claude Code:

Xuru (xuru.ai) — An AI-first support ticket CRM where the AI drafts every response, learns from every resolved ticket, and works through the customer's existing email. No dashboard required. $5 per seat per month, unlimited tickets. In a market where Zendesk charges $55+ per seat.

ProveXa (provexa.ai) — A comprehensive community association management platform with seven specialized AI agents replacing attorney consultations ($0.50 vs. $400 per question), automated meeting minutes from Zoom recordings, true double-entry bookkeeping designed for non-accountants, and resale package generation. $29 per month for what previously required a property management company, an attorney on retainer, and a bookkeeper.

myNewscast — A civic transparency platform that uses AI to make government meetings searchable through automated transcription and processing at $0.40 per meeting, rebuilding the civic content infrastructure that collapsed when classified ad revenue was extracted from local journalism.

These are not prototypes. They are production systems with real unit economics, processing real data, solving real problems at price points that make incumbent solutions look like relics of a different era. They were built by one person with domain expertise and an AI coding partner.

Now multiply this by millions.

III. The Rise of the Builder Class

The dominant narrative about AI and employment is that AI destroys middle-class jobs. This narrative is wrong—not because AI won't displace workers (it will), but because it misidentifies what happens next.

When AI coding tools reach the hands of displaced professionals, those professionals don't become obsolete. They become builders. Their domain expertise—the knowledge accumulated over decades of doing the actual work—becomes the scarcest and most valuable input in software creation. The code is now commodity. The insight is not.

A generic CRM built by an engineering team that interviewed some customers is a fundamentally different product than a CRM built by someone who spent twenty years answering support tickets and knows, in their bones, that small teams live in email and will never log into a dashboard. The AI writes the code. The human provides the judgment that no amount of user research replicates.

This means the coming wave of applications will not be 10,000 copies of the same thing. It will be thousands of applications each serving a niche that was never economically viable for a traditional software company to target—markets too small for venture capital to notice but perfectly real for the domain expert who builds the solution in a few months and charges $29 a month.

The $300 billion SaaS industry was built on the assumption that software is hard and expensive to build. That assumption is now false. What replaces it is not the death of software as a business, but the democratization of software as a creative act—the fulfillment of a promise Steve Jobs once made and then abandoned when he discovered how profitable a walled garden could be.

IV. The Web Wins Again

This wave of applications will be web-native by default. Not for ideological reasons, but for structural ones.

AI coding tools produce web applications. The browser is the universal runtime. Progressive Web Apps have closed the capability gap with native applications. And critically, the volume and velocity of creation will overwhelm any gated distribution model. A single founder building three apps in five months is not going to submit each one to Apple's review process, wait weeks for approval, and hand over 30% of revenue. They're going to deploy to a URL.

The conditions that made the App Store necessary in 2008 no longer exist. Mobile Safari was terrible; now it's capable. Phones were slow; now they're powerful. Offline capability mattered; now connectivity is ubiquitous. The technical justification for the walled garden has evaporated. What remains is pure economic leverage—a tollbooth operator on a bridge that no longer needs a toll.

Meanwhile, Apple's own AI strategy is in disarray. Its AI chief departed after Apple Intelligence was widely criticized. The promised Siri overhaul has been delayed by over a year. Key AI talent is leaving for Meta, OpenAI, and startups. The company is now partnering with Google to provide the intelligence it cannot build itself. The company that controls the most profitable walled garden in history does not control the AI layer that will increasingly determine how users discover and interact with software.

The implication is clear: the gatekeeper's power is shifting from the platform owner to the intelligence provider. The question is whether that shift produces a new open ecosystem or merely a new set of gatekeepers.

V. The Discovery Gap

Protocols like Anthropic's Model Context Protocol (MCP) and Google's Agent-to-Agent protocol (A2A) are building the technical plumbing for a world where AI agents interact with software on behalf of users. These are important and necessary developments. They solve interoperability.

They do not solve discovery.

Consider a concrete example. A small web hosting company in Florida needs a simple, email-native support CRM. The owner asks their AI assistant for a recommendation. The assistant suggests HubSpot, Freshdesk, or Zoho—industry incumbents with brand recognition, extensive review profiles, and deep SEO footprints. It does not suggest the AI-first, email-native solution at $5 per seat that is objectively a better fit for this specific use case. Why? Because that solution was built by one person, has no marketing budget, no review aggregation, no training data presence. The AI assistant—the very technology that enabled the product to exist—cannot find it.

This is not a theoretical problem. It is the central failure mode of the future these AI companies are building. If AI tools enable millions of new applications but AI agents can only recommend the same incumbents they were trained on, the ecosystem stagnates. The builder class creates but cannot reach users. The users get worse recommendations than they should. And the AI companies' own products look less intelligent than they are.

Previous attempts at solving this problem—URL-based app stores, curated directories, review aggregators—have all failed. They failed because they applied a scarcity model to a medium of abundance. You don't browse 10,000 niche applications. You describe a problem and the right one finds you. But for the right one to find you, it must first be findable. The mechanism for that matching does not yet exist in the open ecosystem.

VI. The Trust Problem

Discovery without trust is noise. The implicit value proposition of Apple's App Store and Google Play was never primarily about distribution—it was the promise that "we checked this and it won't steal your data or crash your phone." Remove that, and you return to the chaos of early Windows shareware, except now the stakes involve cloud data, payment credentials, and API access to users' digital lives.

A centralized trust authority is not the answer. The liability is existential—one missed vulnerability, one fraudulent app that slips through, and the

verifier's credibility and business are destroyed. This is the ratings agency problem. Nobody wants to be Moody's.

But trust does not require centralization, and it does not require a single authority. The internet has solved this problem before. SSL certificates, DNS, SMTP—these are open protocols that provide trust and reliability without any single entity bearing total liability. They work because they are standards, not services. They are infrastructure, not products.

What is needed is an approach that separates the inputs of trust evaluation from the act of evaluation itself. Applications should be able to declare, in a standardized and machine-readable format, what they do, what data they access, what services they connect to, and what they cost. AI agents should be able to verify those declarations independently. The protocol provides the X-ray. The AI agent is the doctor reading it. No single entity bears the risk of being the universal authority on trustworthiness.

VII. The Proposal: An Open Application Protocol

We propose—and have built—an open standard that serves as the discovery and trust layer for web-native applications in the age of AI agents. It has three components:

The Manifest

Every participating application hosts a JSON file at a well-known path: /.well-known/oap.json. This manifest declares the application's identity, capabilities, pricing, data practices, security posture, and integration points—all in a structured format designed for AI agent consumption, not human browsing.

The manifest's capabilities section is the most critical. It includes a natural-language summary optimized for semantic matching, a list of problems the app solves (written as a user would describe them), ideal user profiles, and differentiators. When an AI agent needs to match "I need a simple email-based support CRM for my small team" to an application, these fields provide the semantic surface for that match.

The manifest's trust section provides the raw materials for agent-side trust evaluation: what data the app collects, where it's stored, who it's shared with, what authentication methods are used, what external services it connects to. These are declarations, not certifications. Trust is built through consistency between declared practices and observable behavior over time —not through a single gatekeeper's stamp of approval.

The Registry

Discovery requires announcement, not crawling. With billions of domains on the internet, no agent can scan them all looking for manifests. Applications need somewhere to announce their existence. But that somewhere must not become a gatekeeper.

The OAP Registry follows the npm model: anyone can register, no approval is required, and anyone can run their own registry instance. A builder registers by submitting their application's URL. The registry fetches the manifest, validates it against the schema, verifies domain ownership via DNS, checks the application's health endpoint, and indexes it for search. Registration is instant, free, and automated. There is no editorial review, no curation, and no fees.

The critical architectural decision is that the registry stores pointers, not data. All application information lives in the manifest on the app's own domain. The registry caches metadata for search performance, but the source of truth is always the application itself. If any registry disappears, the data survives. Another registry can rebuild its index from any mirror. The /all endpoint makes the complete registry available for anyone to replicate.

This is what separates infrastructure from a gatekeeper. An app store owns your listing. A registry points to your listing. The difference is fundamental.

DNS Verification

Domain ownership is verified via DNS TXT records at _oap.domain.com, following established patterns like DMARC and ACME challenges. DNS serves as the verification layer—not the discovery layer. It confirms that the entity registering an application actually controls the domain. Combined

with health endpoint checks and manifest validation, this provides a baseline of authenticity without requiring human review.

Core Principles

Open and decentralized. No single entity controls registration, verification, or access. Any application can participate. Any AI agent can query any registry. The protocol is a public good, released under CC0.

Machine-readable by design. Application manifests are structured for AI agent consumption. An agent querying the registry can match a user's described need against declared capabilities with semantic precision no app store search has ever achieved.

Trust through transparency, not authority. The protocol provides the inputs for trust evaluation. AI agents perform the evaluation. No single entity bears the existential risk of being the universal trust authority.

Merit-based discovery. The registry surfaces applications based on capability match and verified behavior—not marketing spend, brand size, or training data presence. A $5-per-seat CRM built by one person with deep domain expertise has the same discoverability as a $55-per-seat incumbent with a billion-dollar marketing budget.

Complementary to existing protocols. OAP does not compete with MCP or A2A. It completes them. MCP defines how agents communicate with tools. A2A defines how agents communicate with each other. OAP defines how agents discover and evaluate the applications they connect to on behalf of users. It is the missing layer between "the user has a need" and "the agent knows what exists."

VIII. It Already Works

This is not a proposal for something that should be built. It is a description of something that has been built and tested.

The protocol specification is complete. The manifest schema is defined. Three production applications—Xuru, ProveXa, and myNewscast—have

reference manifests that pass validation. A working registry server accepts registrations, validates manifests, verifies DNS, checks health endpoints, and provides a search API for AI agents. A command-line tool generates manifests interactively in under five minutes. A validator checks any manifest against the spec.

We tested the exact scenario described in Section V: an AI agent queried the registry for “email-native support CRM for small web hosting business.” The registry returned Xuru with a 0.75 match score. The guy in Florida finds the best tool for his needs—not because of brand recognition or marketing spend, but because the application’s declared capabilities match his described problem.

Everything is open source and released under CC0. The registry server is a single Node.js file with a SQLite database. The barrier to running your own registry is effectively zero. The barrier to registering an application is a single API call after deploying a JSON file.

The entire system—protocol, registry, tools, and reference implementations—was built in a single working session using the same AI coding tools that enable the builder class this protocol serves. This is not irony. It is proof of concept at every level.

IX. Why AI Companies Must Lead What Comes Next

A working protocol and a reference registry are necessary but not sufficient. For OAP to fulfill its purpose, AI agents must query it. And the companies building those agents have both the strategic incentive and the practical ability to make that happen.

When a user asks an AI agent for a software recommendation and receives the same five incumbents that a Google search would have returned in 2020, the agent looks stupid. It doesn’t matter how sophisticated the underlying model is. The user’s experience is no better than a search engine. The entire value proposition of an AI agent—that it understands your specific need and finds the best solution—is undermined by the absence of a mechanism to discover what actually exists.

An open registry gives agents access to a structured, searchable, continuously updated index of applications described in terms that enable semantic matching. This makes every agent smarter at the task users increasingly expect them to perform: finding the right tool for the job.

Moreover, each AI company has a strategic interest in this layer being open rather than proprietary. If Google builds a closed app registry, Anthropic and OpenAI will not use it. If OpenAI builds one, Google will not adopt it. A proprietary solution fragments the ecosystem and degrades every agent's capability. An open standard improves them all.

The precedent is clear. Let's Encrypt succeeded because Mozilla, Google, Cisco, and others recognized that universal SSL certification was infrastructure that served everyone's interests. No single company could have done it alone, and no single company needed to own it. OAP requires the same philosophy: industry-backed infrastructure that no single entity controls.

The AI companies did not ask to become the trust layer for software discovery. But they are becoming it whether they choose to or not. Every time an agent recommends a tool, connects to a service, or acts on a user's behalf, the AI company's reputation is at stake. An open protocol with a federated registry is not altruism. It is risk management. And the protocol is ready for them to use.

X. The Economic Counter-narrative

The prevailing narrative—that AI will hollow out the middle class—assumes the current economic structure is permanent. It assumes software will always be built by engineering teams at funded companies, and AI simply replaces those engineers, concentrating wealth further upward.

This assumption is already wrong.

What is actually happening is that domain expertise—the accumulated knowledge of professionals who understand specific problems deeply—is becoming directly convertible into software products without the

intermediation of engineering teams, venture capital, or corporate hierarchies. The value was never in writing code. It was in knowing what to build and why. AI has made the code commodity. The knowledge remains scarce and valuable.

This represents the potential for the largest creation of small business wealth since the early internet. Millions of professionals with deep domain knowledge can now build, deploy, and monetize software solutions for the specific problems they understand better than anyone. They don't need permission from investors. They don't need to hire engineers. They don't need to navigate app store review processes. They need a URL, a payment processor, and a way to be found.

The first two exist. The third now exists as well—if the ecosystem adopts it.

Without open discovery infrastructure, this wealth creation concentrates instead of distributing. Incumbents with existing brand recognition and marketing budgets continue to dominate AI agent recommendations. New builders create superior products that no one can find. The transformative potential of AI coding tools is wasted, and the "AI kills the middle class" narrative becomes self-fulfilling—not because the technology demanded it, but because the infrastructure to support a different outcome was never built.

We have built that infrastructure. The question is whether the ecosystem will use it.

XI. A Call to Build

We have been here before.

In 1995, the web was open and permission-less. Anyone could build, deploy, and reach users. Then economic gravity pulled the ecosystem toward gatekeepers who captured the value without preserving the openness. Classified revenue was extracted from journalism. Software distribution was extracted from the web. In both cases, the technology

existed to maintain an open ecosystem. The infrastructure to sustain it did not.

We are at the same inflection point. AI coding tools are creating a new generation of builders. AI agents are becoming the primary way users discover and interact with software. The protocols for agent communication are being built in the open. But the layer that connects human needs to the applications that serve them—the discovery and trust layer—has been missing.

It is no longer missing. It is built. It is working. It is open.

If we do not establish this as open infrastructure now, it will be built as a proprietary platform. History does not merely suggest this; it guarantees it. Economic gravity always favors gatekeepers unless the infrastructure is deliberately designed to prevent consolidation. OAP is that infrastructure.

To the AI companies: your agents need this. An open, federated registry of structured application data makes your products measurably better at the task users are already asking them to perform. The protocol is built, the registry is running, and integration is straightforward. We invite you to query it, contribute to it, and help establish it as the standard your agents deserve. The specification and all source code are available at oap.dev.

To the builder class—the domain experts who are just beginning to discover what AI coding tools make possible: you are not being replaced. You are being empowered. The knowledge you've accumulated over decades of doing the work is more valuable now than it has ever been. Build your application. Deploy your manifest. Register with the protocol. Be found on merit, not on marketing budget.

To policymakers and the public: the story of AI and employment does not have to be a tragedy. It can be the story of the largest democratization of software creation in history—where the people displaced by AI become the people who build the next wave of applications, serving needs that were never economically viable before. But only if the ecosystem that supports them is designed for openness, not extraction.

The web was supposed to democratize software. For a brief moment, it did. Then the walled gardens rose. We have a chance to get it right this time—not by fighting the gatekeepers, but by building infrastructure that makes gatekeeping irrelevant.

The infrastructure is built. The protocol is open. The registry is running.

Join us.

Appendix A: Technical Overview

Manifest (/.well-known/oap.json)

A JSON file hosted on the application's own domain containing structured declarations across six sections: identity (name, description, URL), builder (creator information and verified domains), capabilities (semantic summary, problems solved, target users, categories, differentiators), pricing (model, starting price, trial availability), trust (data practices, security, external connections, privacy and terms URLs), and integration (API availability, webhooks, import/export capabilities). Full schema available at oap.dev/spec.

Registry (registry.oap.dev)

An open, npm-style index with the following API endpoints:

POST /api/v1/register — Submit a URL for registration. Registry fetches manifest, validates schema, verifies DNS, checks health, and indexes for search. Instant, free, no approval required.

GET /api/v1/search?q={query} — Semantic search across all registered applications. Returns matched apps with relevance scores, trust signals, and pricing. Designed for AI agent consumption.

GET /api/v1/apps/{domain} — Full details for a specific application, including cached manifest, verification status, uptime history, and other applications by the same builder.

GET /api/v1/all — Complete registry dump for mirroring and federation. Enables anyone to replicate the entire index.

The reference implementation is a single Node.js server with SQLite storage. Production deployments can substitute any database and add vector embeddings for improved semantic search.

DNS Verification

Domain ownership is verified via TXT record at _oap.{domain}, following established patterns like DMARC. Format: v=oap1; cat={categories}; price={model}; manifest={url}. DNS serves as verification, not discovery. The registry is the discovery mechanism; DNS confirms the registrant controls the domain.

Tools

oap-init — Interactive CLI that generates a complete manifest through guided prompts. Designed for the builder who has never written a JSON schema and wants to be discoverable in five minutes.

oap-validate — Validates any manifest against the specification. Checks required fields, data types, URL formats, and content quality. Reports errors and warnings with actionable guidance.

Appendix B: Adoption Roadmap

Phase 1: Seed (Current)

Publish the specification, registry, and all tooling as open source under CC0. Deploy manifests to the three reference applications. Launch the public registry at registry.oap.dev. Begin outreach in AI builder communities where Claude Code, Cursor, and similar tools are actively used.

Phase 2: Builder Adoption

Develop framework plugins for Next.js, Remix, and other common deployment targets so manifest generation is a single command. Create integrations with popular AI coding tools so "add OAP to my app" is a recognized prompt pattern. Target 1,000 registered applications across diverse verticals. Publish case studies demonstrating discovery improvements for small builders.

Phase 3: Agent Integration

Engage AI companies to integrate OAP registry queries into their agent recommendation workflows. Demonstrate measurably improved recommendation quality when agents have access to structured application data versus relying solely on training data and web search. Propose OAP as the official complement to MCP and A2A.

Phase 4: De Facto Standard

Sufficient adoption that AI agents check OAP registries by default when users request software recommendations. Multiple competing registries serving different verticals and geographies. Industry governance formalized through a foundation model with backing from AI companies and the builder community, following the Let's Encrypt precedent.

Get Involved

Specification and source code: oap.dev

Registry: registry.oap.dev

Contact: [To be added]

Everything is open. Everything is CC0. Build on it.