Nick Clark · Published April 25, 2026 · PDF

How to Let AI Agents Find and Resolve Each Other Without a Registry

If you are building a multi-agent system, you eventually hit the question of how one agent locates and reaches another without wiring every party to a central directory. This guide walks through an architectural approach for meaning-and-lineage discovery through scoped anchors instead of a central index. It describes an architecture disclosed in United States Patent Application 19/647,395, not a shipping library; the home inventive step is the Semantic Discovery inventive step, and you build it yourself.

What You Are Building

You have a fleet of autonomous agents, or a mix of agents, services, and knowledge objects, and you need one of them to find and reach another at runtime. The naive answer is "stand up a registry": a central service where every agent announces itself and every caller looks others up by name. That works until it does not, and the failure modes are familiar to anyone who has run one at scale: the registry is a single point of truth that must stay synchronized with reality, possessing an address becomes equivalent to having access, and a caller learns nothing about whether the target it resolved is actually the right one for its purpose.

This guide teaches a different shape. It describes how to let agents locate and resolve each other by meaning and lineage, through scoped anchors that each govern their own neighborhood, rather than through a central index. The approach is disclosed in United States Patent Application 19/647,395. It is an architecture you implement, not a package you install.

Why the Obvious Approaches Fall Short

Conventional resolution is lookup-based. DNS, URL resolution, and database key lookups all resolve an address by consulting a table that maps addresses to locations. The table is an external data structure that must be maintained in synchronization with the underlying data. This is a real, well-understood design that works well for its purpose; the point is not that it is broken but that it carries a structural property you may not want in a multi-agent system: the resolver is separate from the thing being resolved, so the two can fall out of sync, and the mapping is a passive answer to a query rather than a governed transaction.

A registry for agents inherits the same shape and adds two more gaps. First, knowing an entry does not tell you whether you are permitted to use it; access control is bolted on beside the lookup rather than being part of it. Second, a flat name or opaque identifier carries no meaning. The caller cannot tell from the address what the target is, what domain it belongs to, or how it relates to anything else. Discovery collapses into "match this string," which is exactly the operation that fails when agents and their neighborhoods change faster than any central catalog can be kept current.

The Architecture

The disclosed approach replaces the central index with a set of governed **anchors** and a persistent traversal entity called a **discovery object**. Every addressable thing in the system, whether content, a knowledge node, a service endpoint, or an execution agent,

is assigned to a nested container governed by an anchor. The anchor is not a pointer; it is an active node that evaluates, filters, and routes entities passing through it.

Addressing is a structured alias, not a flat key. An address takes the form `type@domain.subdomain/path`, for example `agent@net.qu3ry/nest/alpha`, `article@org.wikipedia/computing`, or `dataset@gov.census/2025/population`. This is a human-readable, hierarchically structured address that encodes what the object is, the domain it lives in, and the navigational path through that domain's container hierarchy. It is deliberately not a hash and not a globally unique random string. The meaning is in the address.

Resolution is navigational, not lookup-based. There is no lookup table. A structured alias is resolved by stepwise traversal of the alias path, beginning at the domain anchor that governs the top-level container for the domain and proceeding one segment at a time. At each segment, the current anchor evaluates the next path segment against its own published reachable neighborhood and routes the request to the appropriate sub-anchor, continuing until the terminal segment is reached and the target is returned. Because the alias is resolved by walking the live index that also stores the objects, the resolution always reflects the current state of the structure. There is no separate map to keep synchronized, which is the synchronization problem that lookup-based addressing carries by construction.

Anchors publish their own neighborhoods. Each anchor maintains a dynamic, policy-scoped description of its reachable semantic neighborhood, called the neighborhood publication. Per the disclosure it includes: a semantic content descriptor abstractly summarizing the types and topical character of what is reachable (not an enumeration of every object); a reachability graph of directly navigable sub-anchors and peer anchors with the semantic relationship to each; a policy envelope of the governance constraints that apply to entities traversing the anchor's container; a freshness indicator; and an entropy summary describing the diversity and update frequency of the container. Critically, the publication is maintained by the anchor itself,

not by a central authority, and is recomputed in response to mutation events. This is what lets an agent evaluate where to go next without possessing prior knowledge of the whole index.

Discovery is a persistent object, not a query string. A search, a resolution request, or an agent's reasoning task is instantiated as a discovery object: a persistent, memory-resident semantic entity carrying typed fields. The disclosure lists seven: intent (the structured purpose of the traversal), context (situational parameters), memory (accumulated commitments so far), policy reference (governance constraints that apply), lineage (the ordered record of admitted transitions), affect (modulation parameters), and confidence. The discovery object persists across every step, accumulating state, and it is structurally the same schema as an agent, so the same governance and lineage machinery applies to it without modification.

Every step is a three-in-one traversal step: search, inference, execution. At each anchor the discovery object undergoes three coupled phases. The search step narrows the local neighborhood to a candidate transition set, evaluating the object's current state against what the anchor advertises as reachable; it is local, not a scan of a global index. The inference step scores or selects among those candidates; per the disclosure the inference engine need not be a language model and may be an embedding scorer, a rule matcher, or a probabilistic model. The execution step then decides admissibility, evaluating the proposed transition against policy constraints, lineage continuity, entropy bounds, and temporal validity, and returns one of three outcomes: admit, reject, or decompose into finer sub-transitions. The disclosure frames this separation plainly: the inference engine proposes transitions; the execution substrate decides whether to commit them. Authority to commit resides only in the execution substrate.

Governance travels with the address. Because resolution is a governed traversal, an entity that lacks authorization to traverse a given anchor's container cannot resolve aliases that pass through it, even if it holds the complete alias string. In the disclosure's

terms, possessing an alias does not confer access. This is the property a bolt-on registry cannot give you: the act of finding is the same act as being permitted.

Lineage makes the result auditable. Each transition, whether admitted, rejected, or decomposed, is recorded in the discovery object's lineage field along with the anchor, the state mutation, and the admissibility determination. A resolution is not just a returned address but a walkable record of how it was reached, which is what lets a downstream consumer evaluate whether to trust it.

How to Approach the Build

The following is an ordered way to approach an implementation. The interface sketches below are illustrative and faithful to the disclosure; they are not a library and you implement the mechanisms yourself.

1. **Define the structured alias grammar.** Commit to `type@domain.subdomain/path` and decide your type vocabulary and domain hierarchy. Treat the alias as the primary way anything is named. An agent might be `agent@net.example/nest/alpha`.
2. **Model anchors as containers, not routes.** Each anchor governs a container of objects and sub-anchors, and holds its own mutation policy, alias mapping, and lineage metadata. Give every anchor the ability to recompute and publish its neighborhood. An illustrative shape:

```
Anchor {
  semantic_content_descriptor  // abstracted, not an enumeration
  reachability_graph           // navigable sub/peer anchors + relationshi
  policy_envelope              // who may traverse, what is restricted
  freshness_indicator          // epoch of last publication update
  entropy_summary              // diversity / update frequency
}
```

3. **Make the discovery object the unit of a request.** Instead of passing a query string, instantiate a typed object with intent, context, memory, policy reference, lineage, affect, and confidence. Populate intent and policy from the originating agent's state at initialization; let intent refine as the traversal proceeds.
4. **Implement the three-in-one step as one atomic transition.** At each anchor, run search to produce the candidate set, inference to order it, and execution to admit, reject, or decompose. Do not let inference commit a transition; keep commit authority in the execution substrate so you can swap inference engines freely without weakening governance.
5. **Resolve aliases by walking the same anchors.** Reuse the traversal machinery for alias resolution: start at the domain anchor, evaluate one segment at a time against each anchor's neighborhood publication, and enforce the policy envelope at every hop.
6. **Let anchors self-organize under load.** The disclosure describes anchors splitting a container when entropy or mutation throughput exceeds a policy threshold, merging low-utilization siblings, migrating a container closer to where it is accessed, and rekeying aliases when structure changes, all under deterministic policy and recorded in lineage. When structure changes, keep a policy-bounded redirect chain at the former path so existing references still resolve.
7. **Track drift and record everything.** Maintain a drift metric comparing the discovery object's current intent against its original intent; when it exceeds a policy threshold, flag a drift event and consider re-anchoring intent, backtracking to the last in-threshold step, or reporting the drift alongside the result. Write every admitted, rejected, and decomposed transition to lineage.

What This Does Not Give You

This is an architecture, not a drop-in library. There is no package to install, no SDK, and nothing here "just works" out of the box; you implement anchors, the discovery object, the three-in-one step, alias resolution, and self-organization yourself. The design is disclosed in a patent filing and is educational material, not a benchmarked or production-proven system, and this guide reports no performance numbers because the filing is a disclosure of method and structure.

You still make real engineering choices the disclosure does not make for you: what your inference engine is at each anchor, how your policy envelopes are expressed and evaluated, how anchors are hosted and reached across a network, and how you tune split, merge, and drift thresholds. The approach fits systems where addressing by meaning and lineage and governed access are worth the cost of building governed anchors. If your problem is a small, static set of endpoints that rarely change and carry no access-sensitivity, a plain registry or DNS is simpler and probably the right tool; the value here appears when neighborhoods evolve, access must be intrinsic to resolution, and results must be auditable.

Disclosure Scope

The architecture described in this guide is disclosed in United States Patent Application 19/647,395. This material is provided for educational purposes to explain how the disclosed approach works and how a developer might approach building it. It is not a warranty, a specification of a supported product, or an offer of software, and nothing here should be read as a promise that an implementation will achieve any particular result. Every mechanism described above is drawn from the filing; choices left to the implementer are called out as such.

Semantic Discovery (/semantic-discovery)

All 40 steps → (/inventive-steps)

Search, inference, and execution as one governed step.

Chapter 10 (/patents/19-647395/chapters/discovery)

PRIMARY TECHNICAL DISCLOSURE

- Governed Semantic Discovery: Search, Inference, and Execution Through Adaptive Traversal (/articles/governed-semantic-discovery-search-inference-and-execution-through-adaptive-traversal)

SECONDARY TECHNICAL

- The Adaptive Index as Unified Search-Inference-Execution Substrate (/articles/semantic-discovery/unified-substrate)
- Three-in-One Traversal: Search, Inference, and Execution in a Single Bounded Step (/articles/semantic-discovery/three-in-one-traversal)
- The Discovery Object: A Traversal-Native Semantic Agent (/articles/semantic-discovery/discovery-object)
- Post-PageRank Semantic Ranking: Relevance Through Governed Traversal (/articles/semantic-discovery/post-pagerank)
- Persistent Semantic State: Eliminating Prompt Reconstruction (/articles/semantic-discovery/persistent-state)
- Traversal Lineage as Index Evolution Signal (/articles/semantic-discovery/traversal-lineage)
- Anchor Semantic Neighborhood Publication (/articles/semantic-discovery/semantic-neighborhoods)
- Inference-Time Execution Control as Traversal Primitive (/articles/semantic-discovery/inference-governance)
- Anchor Self-Organization Under Entropy and Load Pressure (/articles/semantic-discovery/anchor-self-organization)
- Alias Resolution as Navigational Traversal (/articles/semantic-discovery/alias-resolution)
- Three Discovery Operating Modes: Human Search, Agent Reasoning, Answer Synthesis (/articles/semantic-discovery/operating-modes)
- Model-Agnostic Semantic Discovery (/articles/semantic-discovery/model-agnostic)
- Affect-Modulated Discovery Traversal (/articles/semantic-discovery/affect-modulated-traversal)
- Confidence-Gated Discovery Traversal (/articles/semantic-discovery/confidence-gated-traversal)
- Integrity-Tracked Traversal Drift Detection (/articles/semantic-discovery/integrity-tracked-drift)

- Biological Identity-Scoped Access During Discovery (/articles/semantic-discovery/biological-access)
- Rights-Grade Anchor Governance for Content Discovery (/articles/semantic-discovery/rights-grade-anchors)
- Forecasting-Shaped Discovery Traversal (/articles/semantic-discovery/forecasting-shaped)
- Capability-Constrained Anchor Accessibility (/articles/semantic-discovery/capability-constrained)
- Collaborative Multi-Object Discovery Traversal (/articles/semantic-discovery/collaborative-traversal)
- Credentialed Reader Activation in the Discovery Substrate (/articles/semantic-discovery/credentialed-reader-activation)
- Personal Cognitive Asset: How Per-User Lineage Re-Weights the Same Substrate (/articles/semantic-discovery/personal-lineage-layer)

APPLICATIONS · GENERAL

- Enterprise Knowledge Management Through Governed Traversal (/articles/semantic-discovery/enterprise-knowledge-management)
- AI-Native Search That Replaces PageRank With Governed Contextual Relevance (/articles/semantic-discovery/ai-native-search)
- Semantic Discovery for Scientific Research (/articles/semantic-discovery/scientific-research-discovery)
- Verifiable AI Legal Case Research: Governed Case Law Search with Citation Lineage (/articles/semantic-discovery/legal-case-research)
- Patent Landscape Analysis Across Classification Boundaries: Governed Semantic Discovery for Prior Art and Freedom-to-Operate (/articles/semantic-discovery/patent-landscape-analysis)
- Governed Medical Literature Search: Evidence-Grade Traversal for Clinical Questions (/articles/semantic-discovery/medical-literature-search)
- Governed Competitive Intelligence Search Across Patents, Filings, and Hiring Signals (/articles/semantic-discovery/competitive-intelligence)
- Governed Semantic Search for Regulatory Compliance: Defensible Requirement Discovery Across Fragmented Corpora (/articles/semantic-discovery/regulatory-compliance-search)
- Cross-Vendor Credentialed Sensor Coordination for Multi-Sensor Perception (/articles/semantic-discovery/coordinated-perception)
- Anti-Stalking Architecture for Cross-Vendor Bluetooth Tracker Networks (/articles/semantic-discovery/post-airtag-tracking)
- How Small Language Models Beat Large Ones: Use the World as Memory (/articles/semantic-discovery/world-as-memory)

APPLICATIONS · SPECIFIC

- Google Search Alternative: Governed Semantic Discovery Beyond Retrieval (/articles/semantic-discovery/google-search)
- Perplexity Alternative: Governed Semantic Discovery Beyond the Answer Engine (/articles/semantic-discovery/perplexity)
- Elasticsearch Alternative: Governed Semantic Discovery Beyond Index-Centric Search (/articles/semantic-discovery/elasticsearch)
- Algolia Alternative: Governed Semantic Discovery Beyond Per-Query Search (/articles/semantic-discovery/algolia)
- Pinecone Alternative for Governed Semantic Discovery (/articles/semantic-discovery/pinecone)
- Weaviate Alternative: Governed Semantic Discovery Beyond the Vector Database (/articles/semantic-discovery/weaviate)
- You.com Alternative: Governed Semantic Discovery Beyond AI Search (/articles/semantic-discovery/you-com)
- Brave Search Alternative: Governed Semantic Discovery Beyond an Independent Index (/articles/semantic-discovery/brave-search)
- Kagi Alternative for Governed Semantic Discovery Beyond Query-Response Search (/articles/semantic-discovery/kagi)
- Exa (Metaphor Systems) Alternative: Governed Semantic Discovery Beyond Neural Link Prediction (/articles/semantic-discovery/metaphor-systems)
- Glean Alternative for Governed Enterprise Discovery: Beyond Stateless Workplace Search (/articles/semantic-discovery/glean)
- Coveo Alternative: Governed Semantic Discovery Beyond Personalized Ranking (/articles/semantic-discovery/coveo)
- Apple Find My vs Governed Semantic Discovery: Two Meanings of Multi-Authority (/articles/semantic-discovery/apple-find-my)
- Google Find My Device Alternative: Governed Cross-Network Reader Activation (/articles/semantic-discovery/google-find-my)
- Governed Discovery Beyond IETF DULT: Behavior Standards vs Architectural Governance (/articles/semantic-discovery/ietf-dult)
- Glean Alternative: Governed Semantic Traversal Beyond Enterprise Search (/articles/semantic-discovery/glean-enterprise-search)
- Microsoft GraphRAG vs Governed Traversal: Adding Per-Step Governance to Graph Retrieval (/articles/semantic-discovery/microsoft-graphrag)
- Governed Agent Memory Beyond Mem0, Zep, and Letta (/articles/semantic-discovery/memory-for-agents)

Semantic Discovery overview → (/semantic-discovery)