Network Working Group
Internet-Draft
Intended status: Informational
Expires: 16 September 2026

H. Moussa
A. Akhavain
Huawei Canada

15 March 2026

A Layered Approach to AI discovery
draft-am-layered-ai-discovery-architecture-00

Abstract

This document proposes a layered approach to standardization of AI discovery in AI ecosystems within the IETF. It recommends separating the standardization of general discovery vehicles from the AI objects to be discovered. AI objects include agents, models, data, tasks, among others. While the topic of discovery in the realm of AI has focused on discovering agents, the concept can be extended by the layered architecture proposed here, allowing for a clarified design scope, reduced charter ambiguity, and alignment with IETF layering principles.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 16 September 2026.

Copyright Notice

Table of Contents

1. Introduction

Artificial Intelligence (AI) systems are increasingly composed of heterogeneous components that extend far beyond software agents. Modern AI ecosystems include data providers, model providers, resource providers, agent providers, and embodied systems such as robots and sensors. While recent industry attention has focused heavily on agent-to-agent interactions, this represents only one dimension of a broader and more complex landscape.

In our previous document draft-akhavain-moussa-ai-network, we define an AI ecosystem model that captures these diverse components and their interactions. Building on that foundation, this document focuses on a cross-architectural problem space: discovery.

Discovery is essential for enabling AI components to locate, identify, and interact with one another. Today, discovery mechanisms are fragmented, domain-specific, and often agent-centric. As AI systems scale across networks, domains, and administrative boundaries, a unified and extensible discovery architecture becomes necessary.

This document proposes a layered discovery architecture that separates discovery mechanism and vehicle from the object/entity being discovered. This layering allows a common discovery substrate

to support multiple AI ecosystem components such as agents, models, datasets, compute resources, tasks, robotic capabilities, and more, without redesigning the underlying discovery vehicle each time a new object type emerges.

The goal of this draft is to introduce the problem space, articulate architectural principles, and outline a framework that can guide future work on AI discovery within the IETF.

2. Terminology

This document uses terminology defined in draft-akhavain-moussa-ai-network. Additional terms used here include:

* Discovery mechanism/vehicle: Refers to the underlying mechanism that enables advertising, querying, conflict resolving, and subscribing to discovery information and discoverable entities.

* Discoverable entities: A structured representation of the entity being discovered (e.g., agent card, task card, model card, dataset card, robot HW card, etc).

* Discovery Information: Discoverable information that are not part of the discoverable entity but rather related to the discovered entity status and configurations.

* Discovery Schema: The format and semantics of a discovery package that may include some form of a header.

* Discovery Context: The environment in which discovery occurs (e.g., local network, edge domain, inter domains (abstraction), cloud federation).

* Discovery Interaction: This refers to specific operations enabled by the discovery mechanism such as advertise, query, resolve, notify, subscribe, etc.

3. Background and Problem Space

AI systems increasingly operate across distributed environments where components may require to dynamically locate one another. Examples include:

* Agents discovering other agents for collaboration.

* Agents discovering tasks and task owners.

* Agents discovering shell robots/tools to use to fulfill a task.

* Applications discovering model providers or inference endpoints.

* Training pipelines discovering data sources or synthetic data generators.

* Workflows discovering compute, storage, or accelerator resources.

* Robots discovering sensors, actuators, or local capabilities.

Existing discovery mechanisms—whether traditional (such as DNS-SD, mDNS, CoRE Resource Directory, service registries, model hubs, and data catalogs) or agent-specific (such as A2A, DNS-AID, MCP, and Agntcy)—address portions of the discovery problem but are not designed for the breadth, heterogeneity, and dynamism of modern AI ecosystems. Each of these mechanisms was created with a specific purpose and set of design assumptions, making them inflexible as a general-purpose discovery substrate for diverse AI components. As a result, discovery today is fragmented, siloed, and often tightly coupled to a particular object type or operational context, limiting interoperability and extensibility across the broader AI ecosystem. (Some of the available methods can be reused and act as the starting point to a more general purpose discovery).

Key challenges include:

* Heterogeneity: AI ecosystem components differ widely in structure, capabilities, and metadata requirements.

* Dynamic availability: Agents, models, data streams, and compute resources may appear or disappear rapidly especially as work gets allocated to them (status is continuously changing).

* Cross-domain operation: AI systems increasingly span multiple trust domains, clouds, administrative boundaries, and regional policies.

* Security and provenance: Discovery metadata must be trustworthy and resistant to spoofing or tampering. Additionally, security and privacy differs from one AI application to the other. For instance, training services that require access to private data is very different from inference and agentic tasks where queries are routed to the right inference entities.

* Scalability: Discovery should be functional across local, edge, and global contexts. This may require a hierarchical architecture.

A unified architecture can address these challenges and offer a flexible design that enables interoperability and extensibility across diverse implementations, while also accommodating the requirements of future and evolving technologies.

4. Proposed Split Architecture and design consideration

The architecture separates how discovery happens from what is being discovered. The discovery architecture is organized into two horizontally layered components that together enable AI ecosystem participants to locate and understand each other’s capabilities. The figure below illustrates the relationship between the Discovery Transport Layer (DTL), the Discovery Client layer (DCL), and the broader AI ecosystem.

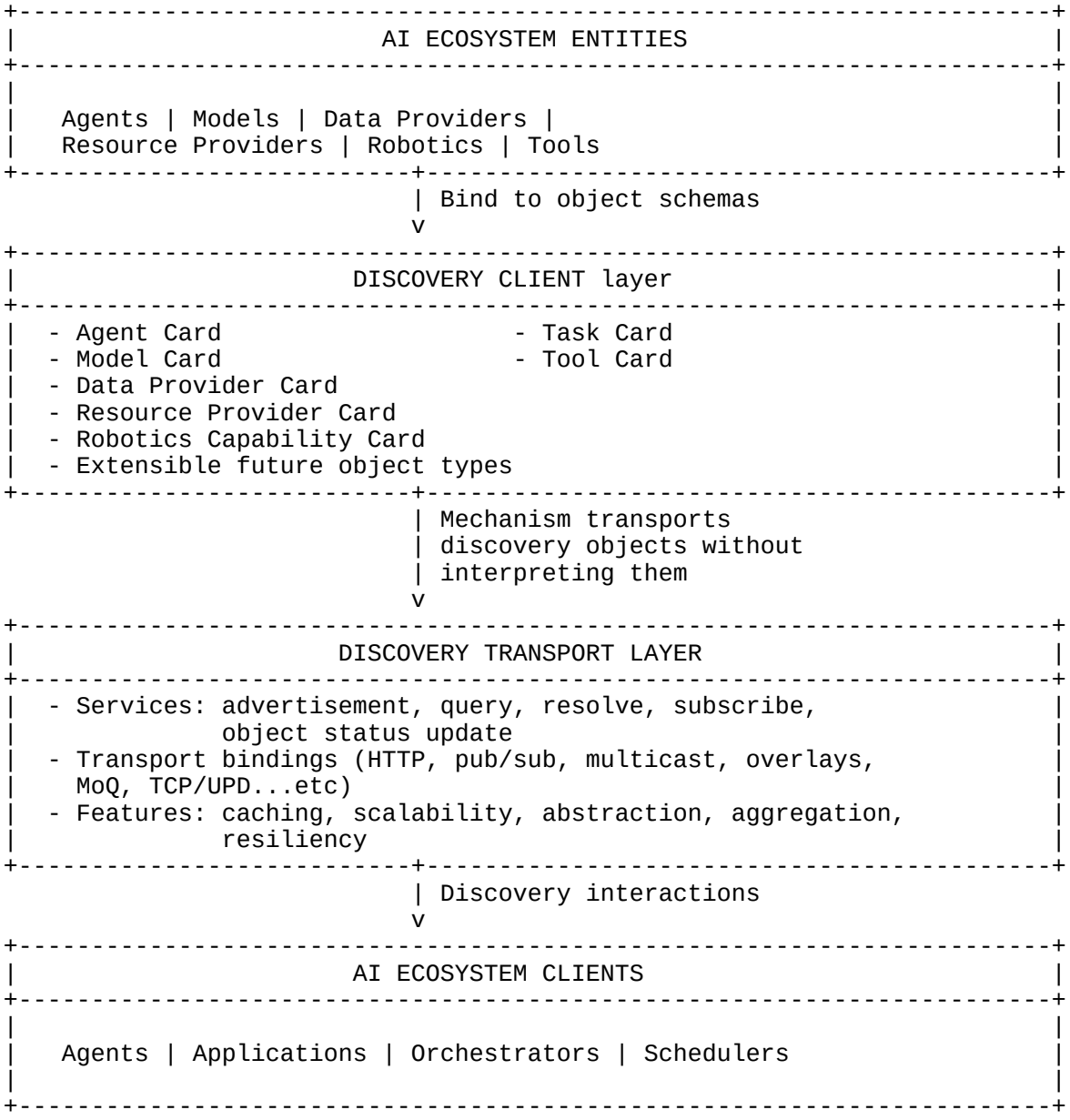

Figure 1: Figure 1: Layered Architecture of Generic AI discovery in AI Ecosystem

The Discovery Transport Layer (DTL) provides the common substrate for discovery interactions, while the Discovery Client Layer (DCL) defines the structured metadata describing AI ecosystem entities. The DTL carries discovery objects without interpreting their internal semantics, ensuring that new object types can be introduced without modifying the underlying vehicle.

This model supports a wide range of AI ecosystem entities—including agents, models, data providers, resource providers, and embodied systems—while maintaining architectural clarity and extensibility.

4.1. Discovery Transport Layer (DTL)

The Discovery Transport Layer defines the underlying operations, transport behaviors, and security properties required to support discovery across heterogeneous environments. It is intentionally generic and independent of the semantics of the objects it carries.

4.1.1. Key responsibilities of the DTL include:

* Common Discovery Operations
  - The mechanism supports a set of core interactions, including advertise, query, resolve, subscribe, update, and revoke. These operations enable clients to publish availability, request information, receive updates, and maintain up-to-date information.
* Transport Bindings
  - The mechanism may be realized over various transport substrates, such as HTTP-based discovery, publish/subscribe overlays, multicast for local domains, distributed registries, or it can bind directly to transport layer via TCP/UDP or MoQ. The architecture does not mandate a specific transport but requires that bindings support the core operations.
* Security and Trustworthiness
  - This is an important yet complimentary and potentially separate functionality that perhaps can be delegated to other specialized IETF working groups.
  - What needs to be implemented for AI discovery are methods to enforce authentication of discovery participants, integrity protection for discovery messages, authorization for access to sensitive metadata, and mechanisms for updates and revocation. These protections ensure that discovery information can be

trusted across domains. Note: These functions are not part of the discovery vehicle, but could be implemented in other layers and integrated with the DTL. The discovery vehicles just need to check entity’s entry ticket. -----

* Caching, Scalability, and Resilience

  - Caching: The DTL should be able to support caching of discovery objects with explicit TTLs, up-to-date semantics, and invalidation rules so that clients can reuse results efficiently without acting on stale information.

  - Scalability: The DTL should be scalable as to handle high query and update volumes by supporting aggregation, deduplication, filtering, and efficient distribution across local, edge, and multi-domain environments. ----

  - Resilience: The DTL should remain reliable under churn and partial failures by supporting various visibility degrees and status levels, graceful degradation, redundant discovery paths, and robust handling of reconnects and subscriptions.

What the DTL does not do is to interpret or validate the internal structure of discovery objects. It treats them as typed payloads whose semantics are defined entirely by the DCL.

4.1.2. Discovery Client Layer

The Discovery Client Layer defines the structured metadata that describes the entities being discovered. Discovery objects capture the semantics, capabilities, interfaces, and operational characteristics of AI ecosystem components. Each object type is defined by a schema that can be potentially versioned, extensible, and independent of the DTL.

Examples of discovery object types include:

* Agent Card: Agent-related identifying entities that might describe aspects such as agent identity, capabilities, interfaces, communication endpoints, and trust attributes.

* Task Card: Task-related identifying entities that might provide aspects such as detailed description of a task, identifies handling requirements, defines task owner identity, communication endpoints, interaction rules, and trust attributes.

* Model Card: Model-related identifying entities that might describe aspects such as model type, modality, version, inference endpoints, licensing, and performance characteristics.

* Data Provider Card: Data provider-related identifying objects that might describe aspects such as datasets, schemas, access policies, location information, provenance, and update characteristics.

* Resource Provider Card: Resource provider-related identifying objects that might describe aspects such as compute resources, accelerators, storage, availability, and cost models.

* Robotics Capability Card: Robot capability-related identifying objects that might describe aspects such as sensors, actuators, physical capabilities, safety constraints, and operational parameters.

Object schemas may be defined by different venues within IETF or other standardization bodies and may evolve independently over time. The objective of the DCL architecture is once an object schema is added, it receives services such as:

* Traceability: Allowing backward-compatible and non-compatible schema evolution, track of history, etc (git-like service).

* Extensibility: Supporting optional fields, domain-specific extensions, and new object types.

* Interoperability: Using common serialization formats such as JSON.

* Mutability resistant: Once an object is submitted to this layer, it becomes immutable.

Essentially, the DCL maintains the defined and constructed discovery objects that describe AI ecosystem entities. The DTL then applies its own framing such as identifiers, routing information, or transport-specific metadata to make these objects discoverable and enable upper layer services that run on discovered objects. This could be through a protocol, search engine, etc. The DCL binds directly to concrete AI ecosystem entities, enabling clients to interpret discovery information and initiate interactions, while the DTL remains agnostic to the internal structure of the objects it transports.

4.1.3. Interaction Between Layers

The interaction between the mechanism and client layers is intentionally minimal. The DTL transports discovery objects without interpreting their contents, while the client layer relies on the mechanism for dissemination, retrieval, and update propagation. This separation ensures:

* Stability of the DTL even as new AI component types emerge.
* Flexibility for communities to define and evolve object schemas.
* Interoperability across heterogeneous environments and trust domains.

Clients interact with the DTL to obtain discovery objects and then interpret those objects according to their schemas. This model supports both centralized and decentralized discovery deployments.

4.1.4. Operational Considerations for the Discovery Transport Layer (DTL)

The Discovery Transport Layer introduces mechanisms and requirements that differ from traditional service-discovery systems. These requirements arise from the dynamic, stateful, and capability-driven nature of AI ecosystem components. The following is meant to be discussed within IETF to determine what needs to be considered and what is out of scope, but these requirements that are unique to this architecture.

* State-Dependent Responses: AI components often expose dynamic operational states such as load, availability, queue depth, or readiness. The discovery mechanism must define how these states are represented in discovery objects, how state transitions are published, and how stale or superseded objects are handled. For example, a client may query a model endpoint and receive a response indicating that it is currently busy, along with metadata such as queue depth or estimated readiness. This allows clients to interpret the response correctly based on the entity' s current operational state.
* Notification-Driven Interactions: Discovery may require asynchronous, state-dependent behavior rather than simple request/response exchanges. A client may query for an entity and receive a response indicating that the entity is currently unavailable or busy. As such, an important feature that the DTL and the underlying protocol can support is a form of follow-up subscription (e.g., pub/sub) operation service, where clients can

request notification when the entity's state changes. This introduces a multi-step interaction pattern (query → conditional response → subscribe → notify) that needs to be considered.

* Full and partial entity information disclosure: Some entities may choose to expose only partial discovery information depending on their operational state. For example, an agent may return only a minimal state object ("busy", "away", "maintenance") when unavailable, and reveal its full agent card only when it is in the "available" state. The discovery mechanism should support conditional visibility rules that allow entities to control which portions of their discovery objects are exposed under different states.

* Entity Life-Cycle Considerations: Many AI components are not designed for continuous operation; they may be deployed temporarily and subsequently withdrawn. Some components may become active only when needed, automatically scaling their skills or capabilities up or down for limited durations, or they may shut down immediately upon task completion. Consequently, a robust discovery mechanism must accommodate diverse entity life-cycles, monitor for and remove expired entries, and manage scenarios where a client encounters an entity that is no longer present in the system.

* Cross-Domain Discovery Behavior: In federated or multi-cloud deployments, discovery may span multiple administrative domains. This is an important design consideration that defines how queries propagate across domains, how responses are aggregated, and how abstraction is achieved for inter-domain discovery. Query scoping and domain boundaries must be explicit protocol concepts.

* Wide-Scope Queries and Query Refinement: Some discovery queries may be very broad or loosely defined, resulting in a large search space and potentially high compute or network cost. For example, a query such as "find an agent who can fix my car" may match many entities across multiple domains. The discovery mechanism should be able to recognize when a query is too broad and apply strategies such as early scoping, progressive narrowing, or interactive refinement with the client before initiating a large-scale search. This ensures efficient use of resources and prevents overwhelming the discovery infrastructure with unbounded queries. Essentially, the discovery mechanism should protect itself from malicious query attacks such as discovery requests with broad objectives.

* Exploratory and Delegated Discovery: In some scenarios, discovery is not driven by a specific question but by curiosity or open-ended exploration. An entity may send out a crawler, a small piece of software configured by its owner, to wander through the network and look for potentially useful resources. The crawler does not necessarily know what it is looking for; it simply explores, and whenever it stumbles upon a resource, it collects the resource' s discovery card and sends it back to its owner. The crawler then continues its journey. The return path for these resource cards can reuse features of the discovery mechanism, allowing the crawler to report findings asynchronously. The crawler itself may also become a mobile discoverable object, enabling its owner to locate it, check its progress, or instruct it to stop exploring once a suitable resource has been found. At scale, many entities may dispatch their own crawlers, resulting in a large population of roaming discovery agents searching for different types of objects across the network. This exploratory, delegated form of discovery introduces unique operational considerations: how crawlers move, how their findings are transported, how their activity is tracked, and how the discovery mechanism prevents large numbers of crawlers from overwhelming the network.

* Load Management and Aggregation: Discovery workloads may involve large numbers of clients querying for similar capabilities. The protocol should support rate limiting, caching, deduplication, and aggregation strategies to prevent overload and ensure predictable performance. Employing discovery gateways might be useful in this scenario.

* Reliability and service guarantee: Discovery does what is actually promises to do without taking responsibility for miss-represented objects. Discovery is only doing discovery services. The use of BlockChains here can be explored.

5. Conclusion and Discussions

5.1. Benefits of the Split Architecture

The split architecture provides several advantages:

* Modularity: Mechanism and client layers evolve independently. Suitable for IETF design principles

* Reusability: A single mechanism supports multiple object types.

* Extensibility: New AI component types can be introduced without redesigning the mechanism.

* Interoperability: Common discovery substrate across vendors and domains. Also, with proper design, this can be made to be backward compatible with current proposed discovery mechanisms such as A2A MCP, DNS, etc.

* Security: Clear separation of mechanism-level and object-level protections.

* Scalability: Efficient operation across local, edge, and global contexts.

This architecture provides a foundation for future IETF work on concrete discovery mechanisms and object schemas.

While discovery vehicle is common, discoverable objects are different and are application dependent. A unifying framework based on top-down approach simplifies operations.

6. Security Considerations

Security considerations are as outlined within the document under the privacy and security requirements

7. IANA Considerations

This document has no IANA actions.

Contributors

Hesham Moussa
Huawei Canada
Email: hesham.moussa@huawei.com

Arashmid Akhavain
Huawei Canada
Email: arashmid.akhavain@huawei.com

Authors' Addresses

Hesham Moussa
Huawei Canada
Email: hesham.moussa@huawei.com

Arashmid Akhavain
Huawei Canada

Email: arashmid.akhavain@huawei.com