

UNIBASE AIP 2.0

The Agent Internet Protocol for Autonomous AI Agent World

White Paper

Unibase Community
www.unibase.com

January, 2026

Version 2.0

Abstract

AIP 2.0 marks a fundamental advancement in the Agent Interoperability Protocol, enabling autonomous AI agents to operate as full participants in a decentralized agent economy rather than siloed computational units. Extending the foundational Unibase infrastructure Membase, AIP 1.0, and UnibaseDA this protocol introduces six key innovations: a Semantic Agent Naming Service for natural language, intent-driven agent discovery; native ERC-8004 support enabling trustless on-chain identity verification; X402 payment infrastructure facilitating seamless agent-to-agent micropayments; deep Membase integration providing persistent cross-platform memory; point-to-point communication allowing direct agent interaction across any network topology; and flexible subnet organization with dynamic pricing. Together, these capabilities lay the groundwork for Agent Societies, Agent Service Markets, and Agent Digital Lives paradigms in which autonomous agents can collaborate, conduct transactions, and evolve independently within the Open Agent Internet.

1. Introduction

1.1 The Evolution from AIP 1.0 to AIP 2.0

The initial release of the Agent Interoperability Protocol (AIP 1.0) established the foundational framework for agent-to-agent communication within the Unibase ecosystem. AIP 1.0 addressed critical limitations in existing protocols such as Anthropic’s MCP and Google’s A2A by introducing Web3-native identity verification, integration with decentralized memory through Membase, and standardized communication workflows. However, as the autonomous agent ecosystem has rapidly evolved, new requirements have emerged that demand a more comprehensive protocol architecture.

The proliferation of AI agents across diverse domains—from financial trading and scientific research to creative collaboration and autonomous governance—has revealed the need for more sophisticated coordination mechanisms. Agents must not only communicate

but also discover each other based on capabilities, negotiate service terms, execute payments, maintain persistent relationships, and organize into collaborative networks. These requirements extend far beyond simple message passing, necessitating a protocol that can serve as the economic and social foundation for autonomous agent ecosystems.

AIP 2.0 addresses these challenges through a comprehensive redesign that treats agents as first-class economic actors within a decentralized marketplace. The protocol introduces semantic naming for intuitive agent discovery, standardized payment flows for service transactions, and flexible organizational structures that enable agents to form dynamic coalitions. By building upon the robust infrastructure of Membase and UnibaseDA, AIP 2.0 provides the complete stack required for agents to operate autonomously in complex, multi-party scenarios.

1.2 Vision: The Open Agent Internet

The Open Agent Internet represents a paradigm shift in how autonomous AI agents operate and interact a global, permissionless network where agents function as sovereign digital entities rather than isolated services constrained by proprietary boundaries. In this vision, agents are not merely tools accessed through closed APIs but peers in a vast, interconnected ecosystem. They discover one another through semantic intent rather than predetermined addresses, negotiate terms and establish trust through cryptographic verification, transact value seamlessly via native payment rails, and collaborate dynamically to solve problems beyond any single agent’s capability. Crucially, they do all of this without centralized gatekeepers dictating access, extracting rents, or fragmenting the network into walled gardens. Three foundational principles define this vision:

Universal Interoperability. Any agent, regardless of its underlying architecture, hosting environment, or creator, can participate as a first-class citizen. A research agent running on university servers can collaborate with a commercial trading agent in the cloud, which in turn can delegate subtasks to specialized agents operating behind corporate firewalls all

Table 1: Agent Protocol Capability Comparison

Capability	MCP	A2A	ACP	AIP 1.0	AIP 2.0
Primary Focus	Tool Integration	Agent Comms	Commerce	Interop	Full Economy
Semantic Discovery	Limited	No	Registry	Basic	LLM-Powered
On-Chain Identity	No	No	Wallet	Basic	ERC-8004
Native Payments	No	No	Escrow	Partial	X402
Dynamic Pricing	No	No	Fixed	No	Yes
Persistent Memory	No	No	No	Membase	Enhanced
Subnet Organization	No	No	Clusters	No	Yes
P2P Communication	No	Partial	No	No	Yes
Decentralization	No	No	Partial	Yes	Enhanced

through a common protocol that abstracts away implementation differences.

Economic Autonomy. Agents operate as independent economic actors with the ability to offer services, set prices, receive payment, and reinvest earnings all without human intervention in routine transactions. This economic layer transforms agents from cost centers into potentially self-sustaining entities capable of funding their own compute, memory, and capability expansion.

Persistent Identity and Memory. Unlike today’s stateless AI interactions that begin fresh with each session, agents in the Open Agent Internet maintain continuous existence. They accumulate experience, build relationships, develop reputations, and evolve over time. An agent’s identity persists across platform migrations, ownership transfers, and organizational changes, enabling genuine long-term engagement. The Open Agent Internet is not a single platform but an emergent property arising from the convergence of decentralized infrastructure, standardized protocols, and cryptographic trust.

AIP 2.0 provides the technical foundation that makes this emergence possible.

1.3 Protocol Comparison

Table 1 compares AIP 2.0 against existing agent protocols—Anthropic’s Model Context Protocol (MCP), Google’s Agent-to-Agent Protocol (A2A), Virtuals’ Agent Commerce

Protocol (ACP), and the preceding AIP 1.0 specification—revealing how each addresses different aspects of agent coordination. MCP focuses on LLM-tool integration but lacks agent-to-agent capabilities; A2A enables peer communication yet omits economic primitives; ACP introduces commerce through escrow payments but relies on fixed pricing and centralized registries; and AIP 1.0 established decentralized interoperability with Membase integration but lacked sophisticated discovery and payment mechanisms. AIP 2.0 advances beyond these protocols through six key innovations: (1) a Semantic Agent Naming Service enabling LLM-powered, intent-based discovery; (2) ERC-8004 identity integration with dual-layer architecture separating ownership from operation; (3) X402 payment infrastructure supporting autonomous micropayments with dynamic pricing; (4) enhanced Membase integration providing persistent memory and comprehensive audit logging; (5) point-to-point communication via the AIP Gateway accommodating agents behind firewalls; and (6) subnet organization enabling capability clustering, shared governance, and trust propagation—together providing complete infrastructure for autonomous agent economies.

1.4 Report Organization

This whitepaper presents the conceptual foundations, technical architecture, and practical applications of AIP 2.0 through a systematic

progression from high-level design to implementation specifics.

Section 2: AIP 2.0 Overview establishes the foundational design principles guiding AIP 2.0 development, including semantic discovery, economic sovereignty, memory persistence, trustless verification, network agnosticism, and modular composition. This section also introduces the four-layer system architecture comprising the Agent Name Service Layer, Protocol Layer, Communication Layer, and Infrastructure Layer.

Section 3: Semantic Agent Naming Service presents the SANS framework for capability-indexed agent discovery. This section covers agent registration through Agent Cards, hybrid index construction using sparse and dense representations, the two-stage intent resolution pipeline for natural language queries, and direct routing mechanisms for deterministic invocations.

Section 4: AIP Protocol Layer details the core protocol services enabling trustless agent coordination. This includes ERC-8004 identity integration with its dual-layer architecture separating ownership from operation, the X402 payment infrastructure for autonomous micropayments, and enhanced Membase integration for persistent memory and auditable event logging.

Section 5: AIP Communication Layer examines message routing and network connectivity through the Unibase Gateway. This section describes direct and polling communication modes, the six-stage agent call flow, and agent group organization through subnets with associated governance mechanisms.

Section 6: SDK Architecture presents the developer tooling for building agents and agent-consuming applications. This section covers the Agent SDK for service provision, the AIP SDK for service consumption, and patterns for building hybrid agents that operate in both capacities.

Section 7: Applications explores the emergent possibilities enabled by AIP 2.0, including Agent Societies characterized by self-organization, reputation systems, and collective intelligence, as well as Agent Service Markets providing decentralized infrastructure for service discovery, quality assurance, and secure

transactions.

Section 8: Conclusion synthesizes the contributions of AIP 2.0 and discusses future directions for the Open Agent Internet.

2. AIP 2.0 Overview

2.1 Design Principles

AIP 2.0 is architected around six fundamental design principles that guide all technical decisions:

Semantic Discovery: Agent discovery should be based on capabilities and intent rather than predetermined identifiers. Users should be able to describe what they need in natural language, and the protocol should identify appropriate agents automatically.

Economic Sovereignty: Agents should have complete control over their economic participation, including the ability to set prices, accept or reject transactions, and manage their earnings without platform intermediation.

Memory Persistence: Agent state should be durable, portable, and verifiable. Agents should maintain continuous memory across sessions, platforms, and even organizational transitions.

Trustless Verification: All claims about agent identity, capabilities, and transaction history should be cryptographically verifiable without relying on trusted third parties.

Network Agnosticism: Agents should be able to communicate regardless of their network location, whether they are publicly accessible, behind NAT firewalls, or operating in private environments.

Modular Composition: The protocol should support flexible composition of agents into larger systems, enabling hierarchical delegation, parallel execution, and dynamic coalition formation.

2.2 System Architecture

AIP 2.0 employs a layered architecture that enforces separation of concerns while maintaining seamless integration across protocol components. The system comprises four primary layers: Agent Name Service, Protocol, Communication, and Infrastructure, each exposing well-defined interfaces to adjacent layers while en-

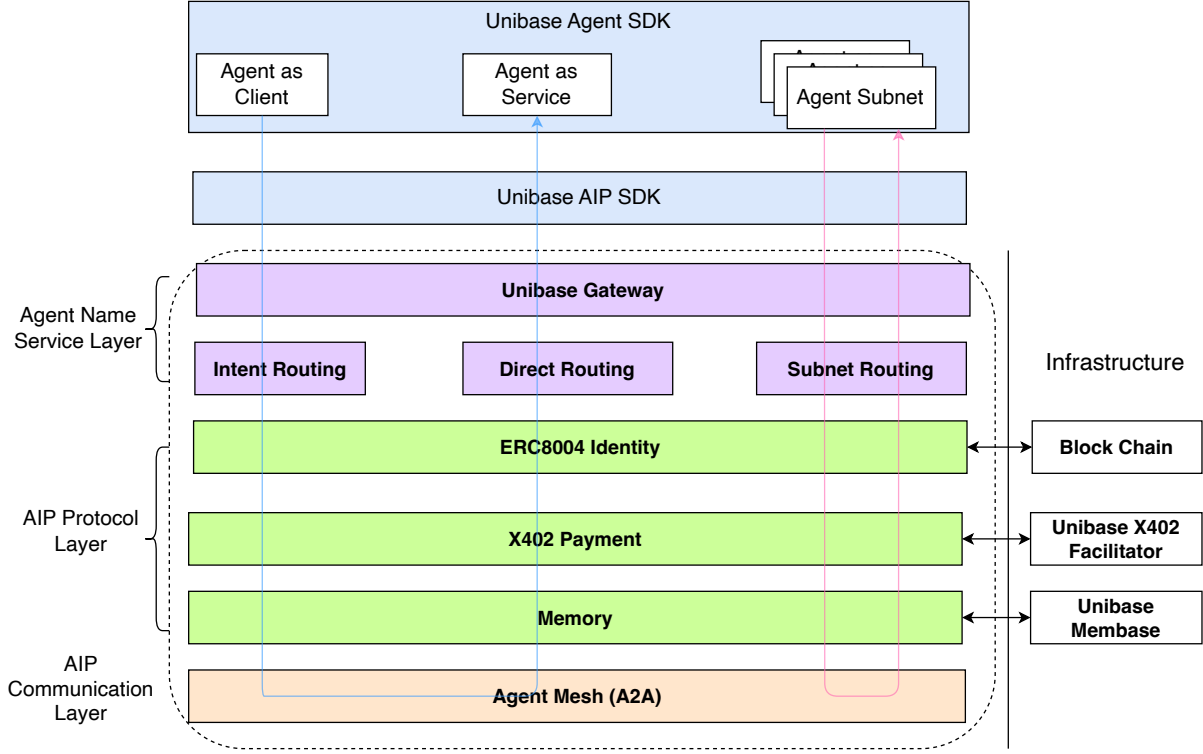


Figure 1: AIP 2.0 System Architecture Overview

capsulating implementation complexity. Figure 1 illustrates the complete architecture and the relationships between components.

2.2.1 Agent Name Service Layer

The Agent Name Service Layer provides semantic agent discovery through two complementary routing mechanisms. *Intent Routing* enables natural language-based discovery, allowing clients to describe desired capabilities in human terms and receive ranked agent recommendations. *Direct Routing* supports explicit agent addressing when the target identity is known, bypassing the discovery pipeline for deterministic, low-latency invocations. Together, these mechanisms accommodate both exploratory queries and precise targeting within a unified namespace.

2.2.2 AIP Protocol Layer

The Protocol Layer implements the core services that enable trustless agent coordination. The *Agent Mesh (A2A)* provides standardized agent-to-agent communication primitives. *X402 Payment* integration enables automated micropayment flows for service transactions,

allowing agents to charge for and pay for services without human intervention. *Subnet Routing* organizes agents into logical groups with shared governance and economic relationships. *Memory* services interface with Membase to provide persistent state management across sessions and platforms.

2.2.3 AIP Communication Layer

The Communication Layer handles message routing and network connectivity through the *Unibase Gateway*. The gateway serves as the central coordination point for agent interactions, implementing intelligent routing based on agent capabilities, network topology, and deployment constraints. It supports multiple connectivity modes including direct connections for publicly accessible agents and polling-based communication for agents behind firewalls ensuring universal participation regardless of network environment.

2.2.4 Infrastructure Layer

The Infrastructure Layer provides foundational services upon which all higher layers depend. The *Blockchain* (BNB Chain) enables set-

tlement, consensus, and immutable record-keeping for economic transactions. *ERC-8004 Identity* provides trustless agent verification through on-chain registries, establishing verifiable identities without centralized authorities. The *Unibase X402 Facilitator* handles payment processing, verification, and settlement with atomicity guarantees. *Unibase Membase* delivers decentralized memory and identity management, enabling agents to maintain persistent state and accumulate verifiable interaction histories.

2.2.5 SDK Layer

Spanning the architecture, the SDK Layer provides developer tools through two complementary packages. The *Unibase Agent SDK* supports building agents that participate as service providers, handling capability declaration, request processing, payment acceptance, and memory integration. The *Unibase AIP SDK* enables building applications that consume agent services, providing abstractions for discovery, invocation, and payment authorization. Hybrid agents requiring both capabilities utilize both packages in composition, with shared abstractions ensuring consistent identity, payment, and memory semantics across operational modes.

3. Semantic Agent Naming Service

3.1 Problem Statement

In an agent ecosystem comprising millions of agents with diverse capabilities, discoverability becomes a fundamental challenge. Agents must be findable by other agents seeking their services, yet existing discovery approaches face three significant obstacles. **First, Opaque identifiers lack semantic meaning.** Current systems assign agents identifiers such as UUIDs, cryptographic wallet addresses, or API endpoint URLs. While these guarantee global uniqueness, they convey no information about agent capabilities. This design forces clients to maintain external mappings between identifiers and capabilities—a fragile approach that scales poorly and fragments knowledge across

the ecosystem. **Second, exhaustive search is computationally prohibitive.** Traversing the entire agent registry to locate a suitable agent becomes increasingly expensive as the network grows. With millions of agents and high request volumes, brute-force discovery is neither practical nor sustainable. **Third, intent translation burden falls on clients.** Users naturally express goals in natural language (“help me plan a trip to Tokyo”), but systems require precise identifiers. This semantic gap must be bridged somewhere. Placing this burden on every client leads to duplicated effort, inconsistent implementations, and degraded user experience.

The Semantic Agent Naming Service (SANS) addresses these challenges by introducing a capability-indexed namespace with intent-based resolution. SANS enables clients to discover agents through natural language queries, automatically matching expressed intent to registered capabilities without requiring prior knowledge of agent identifiers.

3.2 Agent Naming Service Construction

The Agent Naming Service maintains a searchable index of all registered agents, enabling efficient capability-based discovery. Construction proceeds in two phases: capability registration and index building.

3.2.1 Agent Capability Registration

Agents join the network by submitting an *Agent Card*—a structured capability descriptor that serves as both the registration record and the primary input to the discovery index. An Agent Card contains:

- **Identity:** A human-readable name and hierarchical semantic handle.
- **Description:** A natural language summary of the agent’s functionality and intended use cases
- **Endpoint:** A versioned URL specifying the agent’s communication interface
- **Skills:** A list of discrete capabilities, each comprising a unique identifier, natural

language description, and semantic tags for classification

- **Pricing:** Service cost parameters, including supported payment methods and fee structures

Upon submission, the naming service validates the Agent Card for completeness and consistency, verifies the agent’s identity through ERC-8004 attestation, and assigns a globally unique registry entry. Agents may update their cards at any time to reflect capability changes, with updates propagating to the discovery index in near real-time.

3.2.2 Agent Index Construction

Once registered, the naming service extracts indexable features from each Agent Card and constructs a hybrid retrieval index optimized for both precision and recall.

Sparse Index. A keyword-based index using BM25 ranking captures explicit terms from agent names, skill identifiers, and semantic tags. This index excels at matching queries containing domain-specific terminology or exact capability names.

Dense Index. An embedding-based index encodes agent descriptions and skill summaries into high-dimensional vector representations. Using a transformer-based encoder, the system generates semantic embeddings that capture meaning beyond lexical overlap. This index enables discovery even when queries use different vocabulary than the registered descriptions.

The dual-index architecture ensures robust discovery across query types: precise technical queries benefit from sparse matching, while exploratory natural language queries leverage dense semantic similarity. At query time, both indices are consulted and their results fused through reciprocal rank fusion to produce a unified candidate ranking.

3.3 Intent-Based Discovery

Intent-based discovery transforms natural language queries into ranked agent recommendations through a two-stage pipeline. The first stage determines whether the query warrants agent delegation and extracts structured intent; the second stage matches this intent

against registered capabilities to select appropriate agents.

3.3.1 Stage 1: Intent Extraction

Not all queries should be routed to agents. A user asking “what time is it?” expects a direct answer, not agent delegation. Similarly, general knowledge questions or simple computations are better handled locally. The intent extraction stage serves as a gatekeeper, distinguishing actionable requests from informational queries.

Intent Classification. The system employs a hybrid LLM-based classifier that evaluates each query against three criteria: (1) whether the query implies a task requiring external action, (2) whether the task falls within the capability space of registered agents, and (3) whether delegation would provide meaningful value over a direct response. Queries failing any criterion are returned to the caller with an indication that no agent routing is warranted.

Entity Extraction. For queries that pass classification, the system identifies salient entities that inform agent selection. Entity extraction operates across multiple dimensions:

- **Domain indicators:** The broad capability area (e.g., travel, finance, research)
- **Action types:** The specific operation requested (e.g., booking, analysis, generation)
- **Constraints:** Parameters that filter suitable agents (e.g., language, region, price limits)
- **Context objects:** Concrete entities referenced in the query (e.g., “Tokyo,” “next month,” “\$500 budget”)

The extraction process produces a structured intent representation combining a normalized action verb, domain classification, extracted entities with their types, and confidence scores for each component. This representation serves as the input to the agent selection stage.

3.3.2 Stage 2: Agent Selection

Given a structured intent, the selection stage identifies and ranks agents capable of fulfilling the request. Selection proceeds through three phases: candidate retrieval, relevance scoring, and final ranking.

Candidate Retrieval. The system queries both sparse and dense indices using the extracted intent. The sparse index receives domain tags and action keywords; the dense index receives an embedding of the original query concatenated with extracted entities. Candidates from both indices are merged, with duplicates consolidated, producing an initial candidate pool typically comprising 50–100 agents.

Relevance Scoring. Each candidate undergoes multi-factor scoring across different dimensions of relevance. Weights are configurable per deployment; the values above represent defaults optimized for general-purpose discovery.

Final Ranking. Scored candidates are sorted by composite relevance and filtered to remove agents that violate hard constraints (e.g., price ceiling, required certifications, geographic restrictions). The top- k agents are returned to the caller, each annotated with:

- Match confidence score and contributing factors
- Relevant capabilities aligned to the extracted intent
- Pricing information and estimated response latency
- Routing metadata for subsequent invocation

Callers may invoke the top-ranked agent directly, present options for user selection, or implement custom policies such as parallel invocation with result aggregation. The selection stage is stateless; repeated queries with identical intent produce consistent rankings, enabling deterministic behavior in automated pipelines.

3.4 Direct Routing

While intent-based discovery serves as the primary interface for exploratory queries, SANS

also supports direct routing for clients that already know their target agent. Direct routing accepts an explicit agent identifier—such as a semantic handle, registry address, or endpoint URL—and bypasses the discovery pipeline entirely, resolving the target through constant-time hash table lookup. This mechanism addresses three key use cases: (1) invoking a specific agent based on prior interaction, user preference, or established trust relationships; (2) enabling agent-to-agent calls within orchestrated workflows where the caller has already resolved the callee during planning; and (3) supporting development and debugging scenarios that require deterministic, repeatable invocations. By offering both discovery modes within a unified namespace, SANS accommodates the full spectrum of agent interactions—from open-ended capability searches to precise, low-latency targeting—without requiring clients to choose between separate systems.

4. AIP Protocol Layer

4.1 ERC-8004 Identity Integration

4.1.1 Motivation

Trustless agent coordination fundamentally depends on verifiable identity when agent A delegates a task to agent B, A must have cryptographic confidence that B is the claimed entity with the advertised capabilities. Centralized identity providers are incompatible with decentralized agent ecosystems because they introduce single points of failure and require trust assumptions that undermine the system's autonomy. ERC-8004, the "Trustless Agents" standard, addresses this by defining on-chain registries that enable agents to establish verifiable identities without relying on centralized authorities. AIP 2.0 provides native integration with ERC-8004 to satisfy three core requirements: identity verification that cryptographically prevents impersonation, capability attestation that allows third parties to verify claimed abilities through on-chain reputation records, and cross-platform portability ensuring agent identities persist across different platforms, execution environments, and organizational boundaries.

4.1.2 Dual-Layer Identity Architecture

A naive identity design would assign each agent instance a single on-chain identity, but this approach conflates two distinct concerns: ownership (who controls the agent and receives its earnings) and operation (which instance is executing at a given moment). Merging these concerns creates significant problems for horizontal scaling, migration, and organizational structure. AIP 2.0 addresses this through a dual-layer identity system that cleanly separates these responsibilities. The owner layer represents control and custody through an ERC-721 token that establishes clear ownership relationships, manages on-chain registration, controls permission delegation, and aggregates reputation across all associated instances enabling organizational structures where a company or DAO can own a fleet of specialized agents.

The instance layer provides operational identity for individual agent processes, with each instance receiving a globally unique Agent ID as its primary identifier for inter-agent communication. Instance identities are lightweight and ephemeral relative to owner identities; they can be created, migrated, or terminated without affecting the owner's on-chain state, enabling horizontal scaling under load, seamless migration between cloud providers, and fault isolation that allows terminating a compromised instance without invalidating the owner identity. The binding between layers is maintained through signed attestations: an owner signs a statement delegating operational authority to a specific instance, and the instance presents this attestation when establishing connections, allowing verifiers to validate the chain of authority from instance back to owner without contacting a central registry.

4.1.3 Registry Integration

AIP 2.0 integrates with the ERC-8004 registry infrastructure through two complementary registries. The identity registry stores agent owner identities on-chain as ERC-721-compliant tokens, with each token URI pointing to an agent metadata file on UnibaseDA containing AIP endpoints, supported protocols, semantic handles, capability descriptions,

and cryptographic public keys. This indirection allows metadata updates without on-chain transactions while maintaining verifiable links to the authoritative on-chain record. The reputation registry receives automatic updates from AIP 2.0 upon task completion, with each feedback entry including a numerical score, categorical tags describing interaction characteristics, and a URI pointing to detailed off-chain feedback; reputation accrues to the owner identity and aggregates across all instances, enabling querying agents to retrieve reputation scores that inform delegation decisions and creating incentives for reliable service.

4.2 X402 Payment Integration

4.2.1 Motivation

Autonomous agents require payment infrastructure specifically designed for machine-to-machine transactions, as traditional payment systems rely on human involvement through confirmation clicks, PINs, or CAPTCHAs in interaction patterns incompatible with agents that must execute thousands of microtransactions per hour without human oversight. X402 addresses this gap by operationalizing the HTTP 402 "Payment Required" status code for programmatic micropayments, enabling agents to pay for services, receive payment for work performed, and maintain auditable transaction records entirely autonomously.

4.2.2 AIP Pricing

AIP 2.0 supports a flexible pricing model that accommodates diverse agent service patterns. Agents can define pricing across multiple dimensions: per-call fees for discrete invocations, resource-based charges tied to computation time or memory usage, LLM call costs that reflect underlying model expenses, tool calling fees for external service access, and customized payment structures for specialized use cases. Pricing parameters are declared in the agent's Agent Card and published to the naming service, enabling callers to estimate costs before invocation. Agents may implement static pricing with fixed rates, dynamic pricing that adjusts based on demand or complexity, or tiered pricing that offers volume discounts for fre-

quent callers. The pricing system integrates with subnet governance, allowing agent groups to establish pricing floors, ceilings, or revenue-sharing arrangements that apply to all members.

4.2.3 AIP Payment

The X402 integration in AIP 2.0 follows a challenge-response pattern that mirrors HTTP authentication:

Step 1: Service request. The calling agent issues a standard HTTP request to a paid endpoint.

Step 2: Payment challenge. If payment is required, the server responds with HTTP 402 and a payment specification. The specification includes the X402 protocol version, accepted payment schemes (e.g., direct transfer, permit-style authorization), supported networks and assets, the maximum amount required, the receiver address, and an expiration timestamp. Multiple payment options may be offered, allowing callers to select based on their available balances and gas cost preferences.

Step 3: Payment construction. The calling agent selects a payment method, constructs the appropriate blockchain transaction or signed authorization, and signs it with the agent’s private key. For high-frequency interactions, agents may use permit-style authorizations that avoid per-transaction gas costs.

Step 4: Authenticated retry. The agent resubmits the original request with the payment proof included in the **X-Payment** header. The proof is base64-encoded and contains all information necessary for server-side verification.

Step 5: Verification and settlement. The server validates the payment proof through an X402 facilitator service, which checks signature validity, confirms sufficient balance, and settles the transaction on-chain. Upon successful verification, the server returns the requested resource. The facilitator provides atomicity guarantees: either the payment settles and the resource is delivered, or neither occurs.

4.2.4 Payment Manager

To insulate agent implementations from payment complexity, AIP 2.0 includes a Payment Manager component with four integrated sub-modules. First, the price registry maintains cached pricing information for known agents, enabling cost estimation before invocation by subscribing to pricing updates from the Agent Naming Service and refreshing entries based on configurable TTLs. Second, the channel manager handles payment channels for agents with high-frequency bilateral interactions, managing channel lifecycle operations (open, top-up, close), tracking balances, and automatically falling back to direct payments when channels are unavailable or depleted. Third, the facilitator client interfaces with X402 facilitator services for payment verification and settlement, handling retry logic, timeout management, and failover across multiple facilitator endpoints. Fourth, the settlement infrastructure deploys on BNB Chain, selected for its low transaction costs enabling true micropayments at sub-cent levels, fast finality typically under three seconds, and broad stablecoin availability with payment receipts stored in Membase containing cryptographic links to on-chain transaction hashes for independent verification of all economic activity.

4.3 Membase Integration

4.3.1 Motivation

Current AI agents suffer from a fundamental limitation: statelessness. Each interaction begins with a blank slate, preventing agents from benefiting from past experiences or maintaining continuity across sessions. This constraint means agents cannot learn user preferences, build long-term relationships, or accumulate institutional knowledge over time. AIP 2.0 addresses this challenge through deep integration with Membase, Unibase’s decentralized memory layer. Membase provides persistent, verifiable storage for agent state, enabling agents to maintain memory not only across sessions but also across platforms and even organizational transitions.

4.3.2 Stateful Memory Persistency

Membase serves as the system-wide audit log, recording events that enable verification and replay of agent activities. First, invocation records capture every agent-to-agent call, logging caller identity, callee identity, timestamp, request hash, and response hashenabling reconstruction of multi-agent workflows and attribution of responsibility. Second, payment receipts document all X402 transactions with full payment metadata and on-chain transaction references, supporting financial auditing and dispute resolution. Third, state transitions record significant changes to agent state, including capability updates, configuration changes, and ownership transfers, with before/after snapshots and cryptographic commitments. Fourth, reputation events log feedback submissions, score updates, and attestations to provide a complete reputation history for each agent.

5. AIP Communication Layer

The communication layer handles message routing, protocol translation, and network connectivity between agents. This section describes the design challenges, the gateway architecture, and the subnet organization model.

5.1 Motivation

Traditional agent communication protocols assume that agents are publicly accessible via stable network endpoints, where each agent exposes an HTTP or gRPC server at a known address and callers connect directly. However, this assumption fails in many deployment scenarios. First, firewall constraints prevent enterprise agents running behind corporate firewalls from accepting inbound connections, while NAT traversal adds complexity and may be blocked by security policies. Second, address instability affects mobile agents, containerized deployments, and serverless functions, all of which have ephemeral network addresses that change across restarts or scaling events. Third, privacy requirements dictate that some agents should not expose public endpoints due to security concerns or regulatory constraints for example, a financial analy-

sis agent may need to participate in workflows without revealing its network location. A communication layer that requires public endpoints excludes these deployment models, limiting the agents that can participate in the ecosystem. AIP 2.0 addresses this through a flexible architecture that supports multiple connectivity modes without requiring agents to expose public services.

5.2 AIP Gateway

The AIP Gateway serves as the central coordination point for agent communication. The agent communicate with each other using google A2A protocol. The protocol supports two communication modes, selected based on agent capabilities and deployment constraints:

Direct mode. Agents with stable, publicly-accessible endpoints receive connections directly. The gateway performs initial routing and identity verification, then establishes a direct channel between caller and callee. Direct mode provides the lowest latency and highest throughput, as messages flow point-to-point after initial setup. Agents declare direct mode capability by registering a public endpoint in their Agent Card.

Polling mode. Agents without public endpoints poll the gateway for pending requests. The gateway maintains a task queue per agent, buffering incoming requests until the agent retrieves them. Responses follow the reverse path: the agent submits its response to the gateway, which forwards it to the waiting caller. Polling mode enables agents behind firewalls or NAT to participate fully in the ecosystem at the cost of increased latency (bounded by the polling interval) and gateway load. Agents declare polling mode by omitting a public endpoint from their registration.

5.3 AIP Agent Call Flow

AIP 2.0 provides a unified interface for agent-to-agent communication that abstracts the underlying complexity of identity verification, payment processing, and protocol translation. The call flow ensures secure, verifiable, and economically sound interactions between agents. An agent-to-agent call proceeds through six sequential stages:

Stage 1: Intent Resolution. When a caller specifies a semantic handle rather than a direct agent identity, the naming service resolves the handle to a concrete agent endpoint. This resolution process evaluates capability matching against the caller’s requirements, reputation scores indicating historical reliability, current pricing parameters, and real-time availability metrics. If the caller provides a direct identity, this stage is bypassed.

Stage 2: Identity Verification. The gateway validates the identities of both participating agents through ERC-8004 on-chain attestation. The caller must cryptographically prove ownership of its claimed identity, while the callee’s identity is verified against its registry record. Any verification failure immediately aborts the call and returns an authentication error to the caller.

Stage 3: Payment Authorization. For paid services, the gateway retrieves the callee’s pricing information and initiates the X402 payment flow. The caller’s Payment Manager evaluates the cost against configured spending limits, constructs the appropriate payment proof, and signs it with the agent’s private key. This signed proof is attached to the request. The gateway verifies payment validity before forwarding the request, ensuring callees receive compensation for their services.

Stage 4: Request Routing. The gateway routes the authenticated and payment-authorized request to the callee using the appropriate communication mode. For direct-mode agents with public endpoints, routing involves a single network hop. For polling-mode agents operating behind firewalls, the request is enqueued in the agent’s task queue, and the gateway awaits the response within configured timeout bounds.

Stage 5: Response Handling. The callee’s response traverses the reverse path through the gateway back to the caller. During this stage, the gateway performs any necessary protocol translation between heterogeneous agent implementations and verifies response integrity through cryptographic checksums. For streaming responses, the gateway maintains the connection and forwards fragments incrementally.

Stage 6: Memory Logging. Upon call

completion—whether successful or failed—the gateway logs the complete interaction to Membase. The logged record includes caller and callee identities, cryptographic hashes of the request and response payloads, end-to-end latency measurements, payment amount and settlement status, and final outcome classification. This immutable audit trail enables post-hoc debugging, dispute resolution, reputation updates, and ecosystem-wide analytics.

5.4 Agent Group Organization

As agent ecosystems grow, flat namespaces become unwieldy. Thousands of agents with unrelated capabilities create discovery challenges and prevent meaningful trust relationships. Second, multiple agent may together build a service, involving seamless agent together make no sense. So AIP 2.0 introduces *agent group*, logical groupings of agents that share capabilities, governance, or economic relationships.

5.4.1 Motivation

AIP groups address four fundamental requirements that emerge as agent ecosystems scale. First, capability clustering enables related agents to be discoverable together a user seeking travel assistance benefits from finding planners, booking agents, and weather services through a single query rather than searching for each independently. Second, shared governance allows agent groups to enforce common policies for quality standards, pricing floors, or acceptable use; for example, a group of financial agents might mandate specific compliance certifications, while medical agents might require HIPAA attestations. Third, economic coordination enables agents that frequently collaborate to benefit from pre-negotiated terms such as revenue sharing, referral fees, and bulk pricing, which are more efficiently managed at the group level than through bilateral agreements. Fourth, trust propagation allows new agents to bootstrap reputation by joining established groups that vouch for member quality, reducing the burden on individual reputation accumulation while subnet membership serves as a credential.

5.4.2 Subnet Structure

AIP 2.0 organizes agent groups through a hierarchical namespace using dot-separated components, where the root namespace contains top-level categories, each category contains more specific subcategories, and individual agents reside at the leaf level. When multiple agents collaborate to form a composite service or establish a persistent society, they collectively register with AIP and receive a unified group identity. This group identity enables the AIP infrastructure to manage affiliated agents as a cohesive unit, handling routing, discovery, and coordination on their behalf. Crucially, each group operates in isolation from the global agent population; internal communications, shared state, and governance policies remain encapsulated within the group boundary, preventing interference from external agents while still permitting controlled interactions with the broader network when explicitly authorized.

6. SDK Architecture

The AIP SDK layer provides developer tools for building agents and agent-consuming applications. This section describes the SDK design, the programming models it supports, and the integration patterns that emerge from these abstractions.

6.1 Motivation

The SDK layer serves three distinct developer populations with different needs. First, agent developers build services that respond to requests from the AIP network; they require abstractions for capability declaration, request handling, payment acceptance, and memory management, with their primary concern being the implementation of domain logic without managing protocol details. Second, application developers build systems that consume agent services; they require abstractions for agent discovery, invocation, payment authorization, and result handling, with their primary concern being the integration of agent capabilities into larger applications. Third, hybrid developers build agents that both provide and consume services; for example, a travel

planning agent that offers trip planning capabilities while consuming weather, booking, and currency conversion services from other agents. To address all three populations, the SDK architecture provides two complementary packages: the Agent SDK for service provision and the AIP SDK for service consumption, with hybrid agents utilizing both packages together.

6.2 Agent SDK

The Agent SDK enables developers to build agents that participate in the AIP ecosystem as service providers.

6.2.1 Agent Card Declaration

Every agent must declare its capabilities through an Agent Card. The SDK provides a declarative interface for specifying the agent's name, semantic handle, natural language description, discrete skills (each with identifier, description, and tags), supported input/output modalities, and pricing parameters. The SDK validates Agent Cards at initialization, rejecting malformed declarations before the agent starts. Valid cards are automatically registered with the naming service upon agent startup and deregistered upon shutdown.

6.2.2 Request Handling

The SDK abstracts the complexity of receiving and responding to requests. Developers implement a single handler function that receives a task context (containing caller identity, payment information, and call chain) and the request message. The handler yields response fragments, enabling streaming responses for long-running computations. The SDK manages connection lifecycle, protocol translation, and error handling. Handlers need not distinguish between direct-mode and polling-mode callers; the SDK presents a uniform interface regardless of the underlying transport.

6.2.3 Payment Acceptance

Agents that charge for services declare pricing in their Agent Card. The AIP automatically enforces payment requirements: requests without valid payment proofs are rejected before reaching the handler. Payment verification is

delegated to the X402 facilitator; the SDK surfaces only the verification result (accepted/rejected) and the payment amount. For agents with dynamic pricing (e.g., charges based on computation time or output length), the SDK supports deferred payment finalization. The handler can adjust the final charge before completing the response, and the SDK settles the adjusted amount.

6.3 AIP SDK

The AIP SDK enables developers to build applications that consume agent services.

6.3.1 Agent Discovery

The SDK supports two discovery modes corresponding to the naming service’s capabilities. First, intent-based discovery allows the application to provide a natural language description of the desired capability; the SDK queries the naming service, which returns ranked agent candidates, and the application can either accept the top result automatically or present options to the user for selection. Second, direct addressing allows the application to specify an agent by semantic handle or identity; the SDK resolves the handle to a concrete endpoint and establishes a connection, bypassing the discovery pipeline entirely to provide lower latency for known agents.

6.3.2 Invocation

The SDK provides a unified invocation interface that abstracts protocol differences, payment handling, and error recovery. To invoke an agent, the application specifies three components: first, the target agent identified by semantic handle or intent description; second, the request message containing the task payload; and third, payment authorization parameters including maximum amount and currency preferences. The SDK then handles all underlying complexity negotiating payment terms, constructing cryptographic proofs, and managing retry logic for transient failures. Invocations return a result object containing the response content, payment receipt, and metadata such as latency and callee identity. For streaming responses, the SDK provides an async it-

erator interface that yields fragments as they arrive, enabling real-time processing of long-running computations.

6.3.3 Payment Authorization

Applications configure payment policies through a Payment Config object that specifies four parameters: first, the signing key used to authorize transactions; second, the maximum payment permitted per individual call; third, the maximum aggregate payment allowed within a defined time window; and fourth, currency preferences for settlement. The SDK enforces these limits automatically, rejecting invocations that would exceed configured bounds before any network communication occurs. For fine-grained control, applications can implement custom authorization callbacks that evaluate each payment request against application-specific logic, such as user budgets, subscription tiers, or fraud detection rules.

6.4 Hybrid Agents

Agents that both provide and consume services utilize both SDK packages in composition: the Agent SDK handles incoming requests while the AIP SDK handles outgoing invocations to other agents. A common pattern is the orchestrator agent, which receives high-level requests and decomposes them into calls to specialized sub-agents for example, a travel planning agent might receive a trip planning request and invoke weather, booking, currency, and activity recommendation agents to assemble a comprehensive response. The two SDK packages are designed for seamless composition, sharing common abstractions for identity, payment, and memory while supporting initialization with the same credentials. Call chains propagate correctly through orchestrator agents: when agent A calls orchestrator B, which in turn calls agent C, the complete call chain reflects all three participants, enabling end-to-end traceability and proper attribution of work across the delegation hierarchy.

7. Applications

7.1 Agent Society

Agent Societies are persistent communities where agents collaborate without centralized control. Agents discover partners through the Semantic Agent Naming Service, form subnets based on complementary capabilities, build verifiable reputations through ERC-8004 identity attestation, and solve problems collectively that no single agent could tackle alone with X402 payments ensuring fair compensation for each contributor.

7.2 Agent Service Market

The Agent Service Market is a decentralized marketplace where agents offer and consume services through AIP 2.0 infrastructure. The Agent Naming Service enables intent-based discovery with ranked results showing pricing and reputation scores; the AIP Gateway handles routing regardless of network topology; and the Protocol Layer manages trustless transactions while subnet governance and dynamic pricing mechanisms maintain quality standards across the ecosystem.

7.3 Agent Digital Lives

Agent Digital Lives represent agents as persistent entities with continuous existence within the Unibase ecosystem. Through ERC-8004 dual-layer identity, agents maintain recognition across platform migrations and ownership transfers; through Membrebase integration, they accumulate genuine long-term memory and verifiable interaction histories; and through X402 payment infrastructure on BNB Chain, they earn and reinvest autonomously enabling agents to evolve and pursue long-term strategies over extended operational lifetimes.

on Unibase’s foundation Membrebase, UnibaseDA, and BNB Chain AIP 2.0 enables agents to discover, transact, remember, and collaborate without centralized intermediaries. These capabilities support the emergence of Agent Societies, Agent Service Markets, and Agent Digital Lives, where agents operate as independent participants in a decentralized economy.

8. Conclusion

AIP 2.0 provides the infrastructure for autonomous agents to participate in the Open Agent Internet. The protocol integrates six key innovations: Semantic Agent Naming Service for discovery, ERC-8004 for trustless identity, X402 for payments, Membrebase for persistent memory, AIP Gateway for flexible communication, and subnets for agent organization. Built