Internet Engineering Task Force (IETF)
Internet-Draft
Intended status: Standards Track
Expires: October 20, 2026

Y. Wang
HJS Foundation

April 2026

Cognition-Oriented Emergence (COE): A Cognitive Interaction Protocol for Shared World Models

draft-wang-coe-00

Abstract

This document defines the Cognition-Oriented Emergence (COE) protocol, a lightweight, minimalist, narrow-waist cognitive interaction protocol. COE provides a unified "cognitive interaction layer" for the fragmented ecosystem of world models. Through four core primitives -- Judge (J), Delegate (D), Terminate (T), and Verify (V) -- COE enables heterogeneous world models and AI agents to reach verifiable consensus on shared world states within a traceable, auditable, and evolvable framework.

Unlike JEP, which focuses on "accountability tracing" (post-hoc audit), COE focuses on "cognitive consensus" (ex-ante / in-situ collaboration). COE does not depend on any specific world model implementation; instead, it serves as a common "trust operating system" and "collaboration grammar" for them. JEP answers "who is responsible," and COE answers "what the world is" -- both share the J/D/T/V primitive gene, forming a complete "cognition窶疎ccountability" dual-loop architecture.

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 October 20, 2026.

Copyright Notice

Table of Contents

1. Introduction

1.1. Problem Statement

The world model field is flourishing with diverse technical approaches (e.g., JEPA, Dreamer, World Labs, Cosmos). However, these models are isolated from each other in architecture, data format, and interaction style, creating a severe fragmentation problem. When multiple agents (including humans and AIs) need to collaboratively perceive and understand the same physical space, three core problems become unavoidable:

- Conflict Resolution: Observations from different world models may contradict each other; no uniform mechanism exists to reconcile them.
- Consensus Formation: No trusted way exists to agree on "what the world state is at this moment" across multiple observers.
- Traceable Verification: The process by which consensus is reached cannot be traced or audited, eroding the trust foundation.

COE is designed as a standardized cognitive interaction layer to address these three problems.

1.2. Relationship with JEP

COE and JEP share the same J/D/T/V primitives but serve different protocol layers:

- JEP (Judgment Event Protocol): Focuses on accountability tracing (post-hoc audit), answering "who decided what, under whose authority, and who verified it" -- i.e., "who is responsible." JEP follows the philosophy of "minimal intrusion, maximum compatibility."

- COE (Cognition-Oriented Emergence): Focuses on cognitive consensus (ex-ante / in-situ collaboration), answering "how do we collectively know what the world is" -- i.e., "what is the state."

Together they form a complete "cognition窶疎ccountability" dual-loop system: COE establishes a shared understanding of the world; JEP traces accountability for decisions made upon that consensus. COE events may be referenced by JEP as evidence.

1.3. Design Philosophy

COE follows the "minimal intrusion, maximum compatibility" philosophy and adopts a classic "narrow waist" architecture:
- Minimal core: Only J/D/T/V semantics plus lightweight consensus mechanisms.
- Decoupled layers: Heterogeneous world models produce COE events via adapters; upper-layer applications rely only on the stable cognitive interaction grammar.
- Verifiability: All events carry digital signatures and hash linking; the consensus process is fully auditable.

1.4. Terminology

This document uses the following terms:

- Cognitive Unit (CU): Any entity capable of initiating, receiving, or processing COE events (e.g., human user, AI agent, software service). Each CU has a unique identifier (e.g., DID).
- World Model (WM): A model or framework that understands and predicts physical world states. COE does not prescribe its internal implementation.
- Shared World State (SWS): An agreed-upon description of a physical or logical world state reached by multiple CUs via the COE consensus mechanism at a specific point in time.
- COE Event: A digitally signed JSON object initiated by a CU, describing exactly one atomic operation (J/D/T/V) related to a world state.
- COE Adapter: Lightweight software that translates a world model’s internal outputs into standard COE event format.
- Consensus Engine: Core component of a COE implementation that collects J/V events, resolves conflicts according to a policy, and produces the current Shared World State.
- Audit Chain: A tamper-evident chain of COE events linked by cryptographic hashes in chronological order.

1.5. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Protocol Model

2.1. Core Cognitive Primitives

COE inherits the four-primitive model from JEP and HJS, extending it

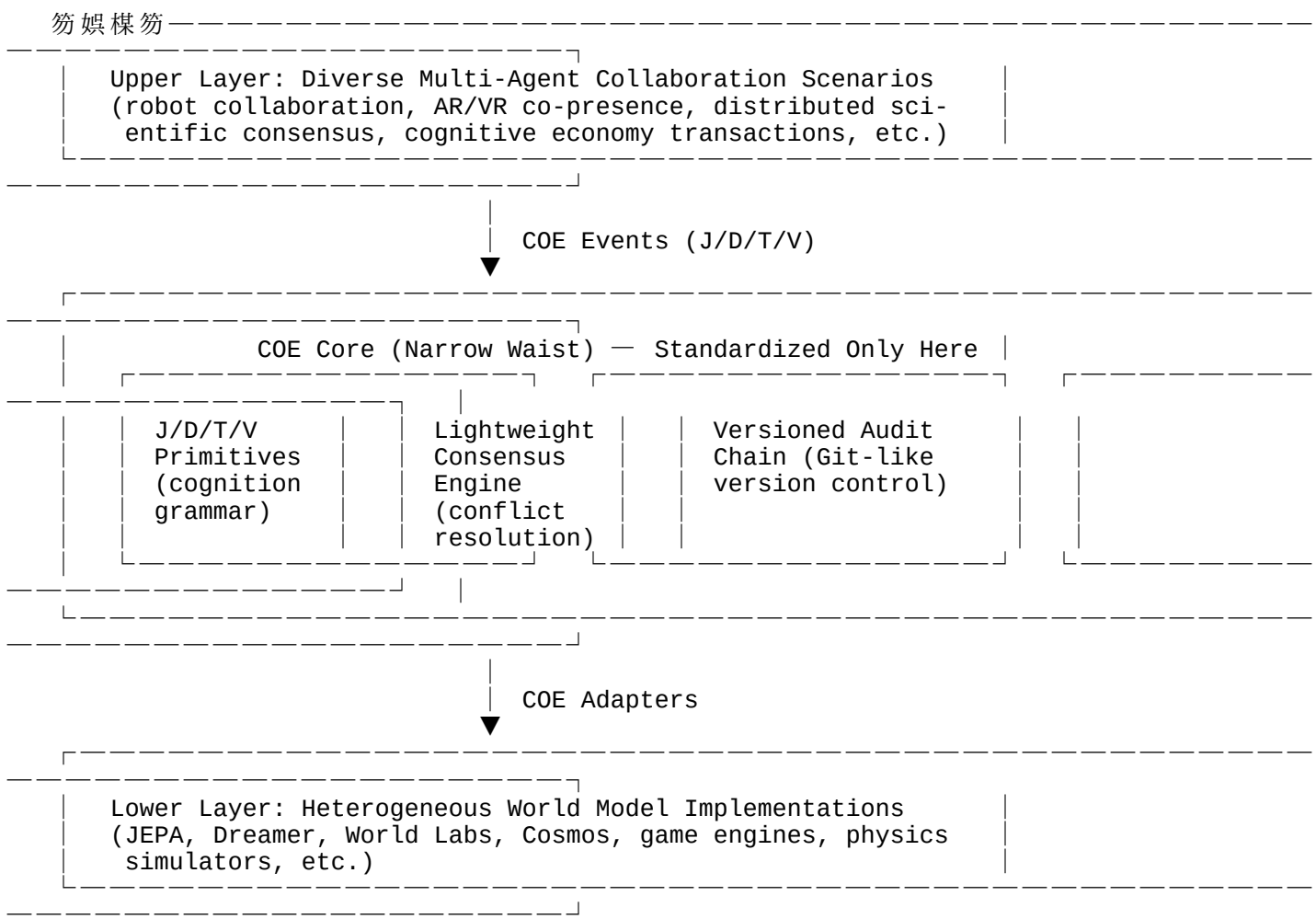
into the "world observation" context:

- J (Judge): Initiates an observation assertion about a world state. A CU declares "I observe X at this time." J events are the starting point of cognitive interaction and the base input for subsequent consensus.
- D (Delegate): Delegates observation or confirmation authority to another CU. This transfers cognitive load, stating "I request / authorize you to observe or confirm X." D events can dynamically adjust the set of consensus participants.
- T (Terminate): Declares that a previously initiated observation assertion (and its associated state) is no longer valid or should be terminated. This closes the lifecycle of a state.
- V (Verify): Cross-validates or confirms one or more observation assertions from other CUs. This is the critical action for consensus emergence, stating "I confirm / verify that observation." V events provide credibility weighting for the consensus engine.

These four primitives form a complete cognitive interaction algebra: any complex multi-agent collaboration scenario can be decomposed into an ordered combination of these four atomic operations. No fifth primitive is required.

2.2. Narrow Waist Architecture

COE adopts a classic "narrow waist" architecture:



Key advantages of the narrow waist:
- Independent evolution: Lower-layer world models can evolve from JEPA to quantum physics engines without changing the COE core;

upper-layer applications can evolve from simple collaboration to global cognitive economy networks while the core protocol remains stable.
- Interoperability: Any world model capable of emitting J/D/T/V events can join the COE network.
- Minimized trust base: The COE core is sufficiently small (reference implementation ~2000-3000 lines of code), making it easy to audit and formally verify.

3. COE Event Format

3.1. General Event Structure

A COE event is a JSON object. All implementations MUST support the following fields:

```
{
  // Required Fields
  "event_id": "<UUID>",
  "protocol": "COE",
  "primitive": "J" | "D" | "T" | "V",
  "issuer": "<CU_ID>",
  "timestamp": "<RFC3339>",

  // World Model Context Fields
  "target": "<WM_ID>",

  // J Primitive Fields (REQUIRED when primitive="J")
  "assertion": {
    "subject": "<OBJECT_ID>",
    "predicate": "<PROPERTY>",
    "value": "<VALUE>",
    "confidence": <0.0-1.0>   // OPTIONAL
  },

  // D Primitive Fields (REQUIRED when primitive="D")
  "delegate_to": "<CU_ID>",
  "delegation_scope": "<SCOPE>",   // OPTIONAL

  // T Primitive Fields (REQUIRED when primitive="T")
  "terminate_of": "<event_id>",
  "terminate_reason": "<REASON>",

  // V Primitive Fields (REQUIRED when primitive="V")
  "verify_of": ["<event_id>", ...],
  "verification_result": "confirmed" | "rejected" | "partial",

  // Audit Chain Fields
  "prev_event_id": "<event_id>",
  "hash": "<SHA256>",
  "signature": "<SIGNATURE>"
}
```

3.2. Field Description

- event_id: Globally unique event identifier; MUST be a UUID v4 [RFC4122].
- protocol: Protocol identifier; MUST be "COE". This distinguishes COE events from JEP or other protocols sharing the primitives.
- issuer: Identifier of the CU that initiated the event; SHOULD be a W3C Decentralized Identifier (DID) [DID-CORE] or other URI.
- timestamp: Event creation time; MUST conform to RFC 3339.
- target: Identifier of the target world model or scene; used to associate all events pertaining to the same physical space.
- assertion: Observation assertion object; REQUIRED when primitive="J".

  - subject: Identifier of the observed object.
  - predicate: Name of the observed property.
  - value: Value of the property; can be string, number, boolean, or JSON object.
  - confidence: OPTIONAL; CU's confidence in the observation, 0.0 to 1.0.
- delegate_to: Identifier of the delegatee CU; REQUIRED when primitive="D".
- delegation_scope: OPTIONAL; describes the scope of delegation (e.g., "continuous_observation", "one_time_confirmation").
- terminate_of: event_id of the J event being terminated; REQUIRED when primitive="T".
- terminate_reason: Reason for termination (e.g., "observation_expired", "state_changed").
- verify_of: Array of event IDs being verified; REQUIRED when primitive="V".
- verification_result: MUST be "confirmed", "rejected", or "partial".
- prev_event_id: event_id of the previous COE event in the chain; may be null for the genesis event.
- hash: SHA-256 hash of the canonical JSON serialization [RFC8785] of the event excluding the signature field.
- signature: Digital signature over the hash field using the issuer's private key; MUST be Ed25519 [RFC8032] or ECDSA with SHA-256.

3.3. Semantic Namespace and Context Isolation

COE and JEP share the J/D/T/V primitives but with different semantics. To avoid conflicts, all COE events MUST include the protocol field with value "COE". Implementations MUST select the appropriate semantic parser based on the protocol field:

- If protocol="COE", interpret according to the "world observation" semantics defined herein.
- If protocol="JEP", interpret according to the "accountability tracing" semantics defined in the JEP specification.

A single CU MAY process both COE and JEP events, provided they are validated within their respective semantic contexts.

3.4. Event Examples

Example 1: Robot A observes that a door is open (J event)

```
{
  "event_id": "550e8400-e29b-41d4-a716-446655440000",
  "protocol": "COE",
  "primitive": "J",
  "issuer": "did:example:robotA",
  "timestamp": "2026-04-19T10:30:00Z",
  "target": "warehouse-zone-3",
  "assertion": {
    "subject": "door_01",
    "predicate": "status",
    "value": "open",
    "confidence": 0.95
  },
  "prev_event_id": null,
  "hash": "sha256:...",
  "signature": "..."
}
```

Example 2: Robot B confirms Robot A's observation (V event)

```
{
  "event_id": "660e8400-e29b-41d4-a716-446655440001",
  "protocol": "COE",
  "primitive": "V",
```

```
  "issuer": "did:example:robotB",
  "timestamp": "2026-04-19T10:30:05Z",
  "target": "warehouse-zone-3",
  "verify_of": ["550e8400-e29b-41d4-a716-446655440000"],
  "verification_result": "confirmed",
  "prev_event_id": "550e8400-e29b-41d4-a716-446655440000",
  "hash": "sha256:...",
  "signature": "..."
}
```

4. Consensus Emergence Engine

4.1. Consensus Engine Overview

The consensus engine is the core component of a COE implementation, responsible for:

1. Collection: Listening for and collecting J and V events targeting the same world model (target).
2. Conflict Resolution: When different CUs issue conflicting assertions about the same subject, resolving the conflict according to the configured consensus policy.
3. Emergence: Producing the current Shared World State (SWS) and recording its generation process.

4.2. Consensus Policies

A COE consensus engine MUST support at least one of the following policies and SHOULD allow pluggable configuration.

4.2.1. Simple Majority Policy

Suitable for scenarios with a small number of CUs and equal trust.

- Rule: Collect V events for a given assertion; the assertion is confirmed when the number of "confirmed" V events exceeds 50% of all V events received for that assertion.
- Convergence condition: Evaluation begins after receiving at least 3 V events; a configurable timeout applies.

4.2.2. Weighted Trust Policy

Suitable for scenarios with heterogeneous CU capabilities and reliability.

- Rule: Each CU is assigned a trust weight w in [0, 1]. For an assertion, calculate confirmed_weight = Σ (w_i * confidence_i) for all confirming V events. The assertion is confirmed when confirmed_weight > threshold.
- Weight management: Weight assignments MUST be managed in an auditable manner (e.g., recorded via JEP events) to prevent centralized manipulation.

4.2.3. Byzantine Fault Tolerance Policy

Suitable for high-security scenarios where malicious CUs may exist.

- Rule: A lightweight BFT variant requiring at least 2f+1 V events (where f is the number of tolerable Byzantine nodes), with more than f+1 being "confirmed".
- Applicability: Recommended when the total number of CUs is at least 4.

4.3. Consensus State Output

Whenever the consensus engine confirms or updates an assertion, it MUST produce a Shared World State (SWS) record:

```
{
  "sws_id": "<UUID>",
```

```
  "target": "<WM_ID>",
  "timestamp": "<RFC3339>",
  "assertions": [
    {
      "subject": "<OBJECT_ID>",
      "predicate": "<PROPERTY>",
      "value": "<VALUE>",
      "confidence": <0.0-1.0>,
      "based_on": ["<event_id>", ...],
      "consensus_policy": "<POLICY_NAME>",
      "confirmations": <COUNT>
    }
  ],
  "previous_sws_id": "<sws_id>",
  "hash": "<SHA256>"
}
```

The consensus state output SHOULD be anchored to the audit chain (see Section 5).

5. Versioned Audit Chain

5.1. Audit Chain Structure

All COE events MUST be cryptographically linked via the prev_event_id field, forming a tamper-evident event chain. Integrity verification algorithm:

```
def verify_chain(events):
    for i in range(1, len(events)):
        assert events[i].prev_event_id == events[i-1].event_id
        computed_hash = sha256(canonical_json(events[i] without signature))
        assert computed_hash == events[i].hash
        assert verify_signature(events[i].signature, events[i].hash, events[i].issuer)
```

5.2. Version Anchoring

Whenever the consensus engine produces a new SWS (Section 4.3), the SWS and all COE events upon which it depends MUST be packaged as a "version" and anchored to the audit chain.

Version record format:

```
{
  "version_id": "<UUID>",
  "sws": { ... },                          // SWS from Section 4.3
  "dependent_events": ["<event_id>", ...],
  "anchor_event_id": "<event_id>",         // Position in the audit chain
  "timestamp": "<RFC3339>",
  "previous_version_id": "<version_id>",
  "hash": "<SHA256>"
}
```

5.3. Timestamp Anchoring

For high-security scenarios, COE implementations SHOULD support submitting version hashes to a public trusted timestamp service (e.g., RFC 3161) or a blockchain (e.g., via OpenTimestamps). Timestamp anchoring provides an independent time proof, further strengthening non-repudiation.

6. Security Considerations

6.1. Threat Model

COE assumes the following threat model:

- An attacker may control a subset of CUs and attempt to inject false observation assertions.
- An attacker may attempt to tamper with published event records.
- An attacker may attempt denial-of-service by flooding events.
- An attacker may attempt to collude to manipulate consensus outcomes.

COE does not assume full trust among CUs, nor does it assume infallibility of underlying world models.

6.2. Integrity Protection

Each COE event contains a hash of its content and the issuer's digital signature. Any tampering with event content will invalidate the hash and thus the signature. The audit chain's hash linking ensures event order cannot be altered. Implementations MUST verify signatures and hashes upon event receipt.

6.3. Sybil Attack Defense

A Sybil attacker may create many fake CU identities to manipulate consensus. Defenses:
- Implementations SHOULD require CUs to undergo trusted registration (e.g., endorsement mechanisms in DID methods).
- Under weighted trust policy, newly registered CUs SHOULD start with low weight, increasing only with demonstrated trustworthy behavior.
- Byzantine fault tolerance policy inherently resists Sybil attacks (provided the attacker controls no more than f nodes).

6.4. Replay Attack Defense

An attacker may capture valid COE events and replay them later. Defenses:
- Each event contains a unique event_id and timestamp. Implementations MUST maintain a cache of processed event IDs and reject duplicates.
- Implementations SHOULD reject events with timestamps deviating excessively (e.g., more than 5 minutes).

6.5. Consensus Blocking Attack

Malicious CUs may deliberately withhold V events or emit contradictory V events to block consensus. Defenses:
- Consensus engine MUST implement a timeout mechanism, ignoring unresponsive CUs after timeout.
- For contradictory V events, the consensus engine resolves according to policy (e.g., majority or weight) and does not stall due to individual malicious CUs.

6.6. Privacy Protection

The assertion field may contain sensitive information (e.g., precise location data). Implementations MAY support selective encryption of the assertion field, making it readable only to designated verifiers holding the decryption key. Hybrid encryption schemes (e.g., ECDH + AES-GCM) are recommended.

Furthermore, COE's three-layer privacy architecture (consistent with JEP) allows implementations to place sensitive payloads in a private payload layer, exposing only necessary public governance information to the audit chain.

7. IANA Considerations

This document may require IANA to establish the following

registries:

7.1. COE Primitive Registry

- Registry Name: COE Primitives
- Registration Policy: Specification Required [RFC8126]
- Initial Contents: J (Judge), D (Delegate), T (Terminate), V (Verify)

7.2. COE Consensus Policy Registry

- Registry Name: COE Consensus Policies
- Registration Policy: Expert Review [RFC8126]
- Initial Contents: simple_majority, weighted_trust, bft

8. Implementation Guide and Reference Implementation

8.1. Minimal Viable Implementation (COE-Lite)

To facilitate community validation and adoption, a minimal reference implementation, COE-Lite, is recommended:

- Core functionality:
  * J/D/T/V event generation and verification (including Ed25519 signatures)
  * In-memory audit chain maintenance
  * Simple majority consensus engine
  * HTTP REST API interface
- Code size: Approximately 2000-3000 lines (Python or Go)
- Excluded: Actual world model integration, persistent storage, distributed communication

COE-Lite aims to demonstrate protocol feasibility, not production-grade readiness.

8.2. Verification Scenario

The following scenario verifies the correctness of COE-Lite:

Scenario: Three simulated robots (A, B, C) collaboratively confirm the state of a door.

1. A issues J event: assertion={door: open, confidence: 0.9}
2. B issues V event: verify_of=[A.J], verification_result="confirmed"
3. C issues V event: verify_of=[A.J], verification_result="confirmed"
4. Consensus engine (simple majority) outputs SWS: door=open
5. A observes door closed; issues T event (terminate_of=A.J), then new J event: door=closed
6. B and C confirm the new state sequentially
7. Consensus engine updates SWS: door=closed

Verification checkpoints:
- Audit chain integrity (all events correctly hash-linked)
- Signature verification (all signatures valid)
- Consensus convergence (both state transitions correctly output)
- Version traceability (complete evolution from open to closed queryable)

8.3. Integration with Existing Standards

- Identity: W3C DID [DID-CORE] or OAuth 2.0 [RFC6749] RECOMMENDED for CU identity management.
- Environmental Trust: RATS [RFC9334] MAY be integrated to attest that a world model runs in a trusted execution environment.
- Accountability: COE events MAY be referenced as evidence in JEP events, enabling seamless linkage from cognitive consensus to

accountability tracing.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, January 2017.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, June 2020.

[RFC9334] Birkholz, H., et al., "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, December 2022.

9.2. Informative References

[DID-CORE] Sporny, M., et al., "Decentralized Identifiers (DIDs) v1.0", W3C Recommendation, July 2022.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[JEP] Wang, Y., "JEP: A Judgment Event Protocol", draft-wang-jep-judgment-event-protocol, March 2026.

[HJS] Wang, Y., "HJS: An Accountability Layer for AI Agents", draft-wang-hjs-accountability, February 2026.

[AIGA] Aylward, E., "AI Governance and Accountability Protocol", draft-aylward-aiga, January 2026.

[MACP] Li, X., et al., "Multi-agent Collaboration Protocol Suite", draft-li-dmsc-macp, February 2026.

Appendix A. Complete Verification Workflow Example

This appendix presents an end-to-end example with a complete COE verification workflow, suitable as a reference test case for proof-of-concept implementations.

A.1. Initial Setup

- CU-A: Robot A (JEPA world model, trust weight 0.9)
- CU-B: Robot B (Dreamer world model, trust weight 0.8)
- CU-C: Human supervisor (trust weight 1.0)
- Target: warehouse-zone-3
- Consensus Policy: weighted_trust (threshold=1.5)

A.2. Event Sequence

```
Step 1: CU-A observes door is open
Event#1 (J): A -> COE: { assertion: {subject: "door",
                       predicate: "status", value: "open",
                       confidence: 0.95} }

Step 2: CU-B delegates continuous observation to CU-A (optional,
        demonstrates D primitive)
Event#2 (D): B -> COE: { delegate_to: "A",
                       delegation_scope: "continuous_observation" }
```

```
Step 3: CU-B confirms A's observation
Event#3 (V): B -> COE: { verify_of: [Event#1],
                     verification_result: "confirmed" }

Step 4: CU-C (human) also confirms A's observation
Event#4 (V): C -> COE: { verify_of: [Event#1],
                     verification_result: "confirmed" }

Consensus calculation:
- confirmed_weight = 0.8 (B) + 1.0 (C) = 1.8 > threshold 1.5
- Output SWS#1: door = "open"

Step 5: CU-A observes door is closed
Event#5 (T): A -> COE: { terminate_of: Event#1,
                     terminate_reason: "state_changed" }
Event#6 (J): A -> COE: { assertion: {subject: "door",
                     predicate: "status", value: "closed",
                     confidence: 0.95} }

Step 6: CU-B confirms new state
Event#7 (V): B -> COE: { verify_of: [Event#6],
                     verification_result: "confirmed" }

Step 7: CU-C confirms new state
Event#8 (V): C -> COE: { verify_of: [Event#6],
                     verification_result: "confirmed" }

Consensus calculation:
- confirmed_weight = 0.8 + 1.0 = 1.8 > 1.5
- Output SWS#2: door = "closed"
```

A.3. Verification Checkpoints

1. Audit chain integrity:
   - Verify all prev_event_id links are correct
   - Verify all hash calculations
   - Verify all signatures are valid

2. Consensus correctness:
   - Verify SWS#1 is based on Event#1, confirmed by Event#3 and Event#4
   - Verify SWS#2 is based on Event#6, confirmed by Event#7 and Event#8
   - Verify the door state transition from open to closed is correctly recorded

3. Version traceability:
   - Query history for "door" should return two versions
   - Each version can be traced back to its dependent J/V events

4. Fault injection tests:
   - Inject event with invalid signature; verify rejection
   - Inject event breaking hash chain; verify detection
   - Inject contradictory V events; verify consensus resolves correctly per policy

Author's Address

Yuqiang Wang
HJS Foundation Ltd.
Email: yuqiang@humanjudgment.org