Internet Engineering Task Force
Internet-Draft
Intended Status: Standards Track
Expires: 29 November 2026

T. Sato
MyAuberge K.K.

29 May 2026

Multi-Agent Delegation in Sovereign Object Systems
draft-sato-soos-mad-01

Abstract

When a consequential task requires multiple AI agents -- one to coordinate, others to execute, each operating on different objects in a shared workflow -- who is responsible for the outcome? Which agent caused which state change? Under whose authority? If the coordinating agent's authorization is revoked, does the authority of every sub-agent it delegated to immediately expire? If one agent in a parallel workflow exceeds its scope, can that excess propagate to others?

Today, no protocol ensures the answers to these questions are knowable. Multi-agent AI systems can produce accountability black holes: chains of delegation where the authority at each hop is unclear, where revocation does not cascade reliably, and where the audit record cannot reconstruct which agent caused which outcome under which authorization.

This document defines the Multi-Agent Delegation (MAD) protocol: a normative specification ensuring that authority in multi-agent AI workflows is structured, verifiable, and reconstructable. MAD specifies three mechanisms. The Narrowing Property (INV-4) ensures that authority can only attenuate across a delegation hop -- a sub-agent can never acquire capabilities its orchestrator does not itself hold. SO Instance Topology Types provide five formally defined patterns for how governed objects relate at runtime. SO Cluster Coordination Primitives (L1-16) enable parallel fan-out topologies in which multiple specialist agents execute simultaneously, with aggregation rules that let the orchestrator proceed as soon as a quorum of specialists complete.

For human principals, MAD provides a single recoverable property: the accountability chain is always reconstructable from the GEC-signed audit record alone. Cascade revocation means one decision stops the entire tree. For AI agents, MAD provides machine-speed delegation authority and the parallel execution efficiency that sequential single-agent approaches cannot achieve.

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 24 November 2026.

Copyright Notice

Table of Contents

1. Introduction

A consequential workflow often requires more than one AI agent. A travel itinerary spanning eight suppliers -- flights, ground transfers, accommodation, activity operators -- may require eight

specialist agents, each authorized to manage one supplier's state, coordinated by an orchestrating agent tracking overall progress. A network management operation may require a coordinating agent that delegates segment-specific routing decisions to specialist sub-agents, each operating within a defined traffic domain. A legal document workflow may fan out to jurisdictional specialists that each produce a clause, then aggregate into a finalized agreement.

Without a delegation governance protocol, these multi-agent workflows produce accountability black holes. Which agent caused which state transition? Under whose authority? If the orchestrator's mandate is revoked -- because a compliance threshold is breached, because a human principal withdraws authorization, because the mission governing the session enters a terminal state -- does that revocation immediately reach the specialist agents it delegated to? If a specialist agent attempts to act beyond its authorized scope, does the confused deputy vulnerability allow that excess to propagate? Without protocol-level answers to these questions, multi-agent AI systems cannot be audited, safely revoked, or relied upon for consequential deployment.

For AI agents, a governed delegation model is not only a safety property -- it is an efficiency property. An orchestrator operating under MAD can delegate to specialist sub-agents at machine speed, without a human bottleneck at each hop, because the Narrowing Property pre-verifies that authority flows only downward. Parallel fan-out topologies allow multiple specialists to execute simultaneously rather than sequentially. Quorum-based aggregation rules let the orchestrator proceed as soon as enough specialists complete, without waiting for the full set. The cluster coordination primitives in this document are the mechanism by which multi-agent workflows achieve the computational efficiency that single-agent sequential approaches cannot match.

MAD addresses these requirements through three complementary mechanisms:

(1) The Narrowing Property (INV-4): a mandate issued to a sub-agent MUST contain only a strict subset of the Cedar actions available to the issuing agent. Authority can only attenuate, never amplify, across a delegation hop.

(2) SO Instance Topology Types: five formally defined patterns describing how multiple Sovereign Object instances relate to each other at runtime, enabling orchestrators and the GEC to reason about multi-agent workflows at the structural level.

(3) SO Cluster Coordination Primitives (L1-16): a GEC-level service for declaring, managing, and querying collections of related SO instances executing in coordination, with cluster-enriched Cedar evaluation and aggregation rules for parallel fan-out patterns.

This document specifies all three mechanisms as a unified Multi-Agent Delegation protocol. It is intended as a companion to draft-sato-soos-aep (Agent Execution Protocol), which defines the per-agent execution loop; draft-sato-soos-sov (Sovereign Object), which defines the SO structure and lifecycle; draft-sato-soos-mjwt (Mandate JWT), which defines the delegation credential format; and draft-sato-soos-hem (Human Escalation Mechanism), which defines how human oversight integrates into multi-agent sessions. MAD is the coordination governance layer across the four-draft stack: IDP [I-D.sato-soos-idp] provides

the per-transition audit artifact at each delegation hop; HEM [I-D.sato-soos-hem] is the escalation mechanism when a hop requires human judgment; GAR [I-D.sato-soos-gar] is the permanent audit record for the full workflow; CAP [I-D.sato-soos-cap] is the prohibition floor applying to every agent at every delegation level.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals.

2. Terminology

The following terms are used in this document. Terms defined in draft-sato-soos-sov and draft-sato-soos-aep apply when used here.

Sovereign Object (SO)
: The unit of governance in SOOS. A causally ordered, policy-governed, living typed document that evolves through a predefined finite state space under GEC-enforced authority.

Mandate JWT
: An Ed25519-signed JSON Web Token granting a specific agent authority to perform specific Cedar actions on a specific SO instance, issued by a principal in the Party Registry.

Narrowing Property
: The invariant that a child mandate’s Cedar action set is always a strict subset of the issuing agent’s own Cedar action set. Defined normatively as INV-4.

Party Registry
: The GEC-managed registry of all principals (operators, agents, humans) and their Ed25519 public keys, and mandate issuance relationships.

Orchestrator Agent
: An agent that coordinates a multi-agent workflow, issuing mandates to specialist sub-agents and managing aggregate progress across multiple SOs or SO Cluster members.

Specialist Agent
: An agent operating under a mandate issued by an orchestrator, with authority narrowed to a specific SO instance and Cedar action subset.

SO Cluster
: A GEC-managed collection of related SO instances executing in coordination. A cluster is a coordination and visibility overlay; it does not itself hold Cedar-governed state.

Cluster Registry
: A GEC-maintained in-memory index of all declared clusters and their member SOs, rebuilt from the Event Log on kernel restart per INV-14.

Aggregation Rule
: A declared condition on SO Cluster member states that, when satisfied, causes the GEC to fire a CLUSTER_AGGREGATION_CONDITION_MET ProximityEvent to the orchestrator session.

Creation Principal Class
: The class of principal authorised to create an SO instance:

HUMAN_DIRECT, AGENT_DELEGATED, or AGENT_AUTONOMOUS.

Mandate Issuance Tree
: The directed tree of mandate issuance relationships maintained in the Party Registry, used to compute CASCADE_TO_DESCENDANTS revocation scope.

GEC (Governance Execution Controller):
: The runtime component that enforces MAD coordination primitives. A GEC maintains the Cluster Registry, enforces INV-4 at mandate issuance, executes Cedar policy at each transition, records all events to the GEC-signed Event Log, and fires ProximityEvents to orchestrator sessions. Earlier versions used "kernel" for this component; GEC is the normative term across the SOOS stack.

GEE
: Goal Execution Engine. An optional SOOS OS service that inverts control, driving the agent execution loop on behalf of an orchestrator rather than the agent driving its own loop.

3. Multi-Agent Mandate Model

3.1. The Narrowing Property

INV-4 (Narrowing Property) is the foundational invariant of the SOOS multi-agent delegation model.

INV-4: A Cedar action MUST only appear in a mandate if it is a subset of the issuing agent’s own Cedar action set. The Narrowing Property MUST be enforced at mandate issuance, not only at evaluation.

This invariant has three consequences:

(a) Authority can only attenuate across a delegation hop. A specialist agent cannot acquire capabilities its orchestrator does not itself hold. An orchestrator cannot grant what it does not have.

(b) The confused deputy attack is structurally prevented at the mandate layer. A malicious or compromised sub-agent that attempts to invoke actions beyond its mandate will be rejected at Step 1 (Mandate Validation) of the kernel execution sequence defined in draft-sato-soos-aep Section 4.2.

(c) Revocation of an orchestrator mandate cascades to all descendant mandates in the issuance tree (Section 3.5).

Implementations MUST enforce INV-4 at mandate issuance time in the Party Registry, not solely at gec.transition() evaluation time. A mandate that violates INV-4 MUST be rejected by the Party Registry before it is issued.

3.2. Mandate Issuance Tree

The Party Registry MUST maintain a mandate issuance tree: a directed tree of mandate issuance relationships where each node is a Mandate JWT and each directed edge records that the parent mandate was used to issue the child mandate.

The mandate issuance tree MUST record, for each issued mandate:

parent_mandate_jti
: The jti of the mandate used to authorise this issuance. NULL for mandates issued directly by a human-held Party Registry principal.

issuing_principal	The Party Registry ID of the issuing agent or human.
cedar_action_set	The Cedar actions granted. MUST satisfy INV-4 with respect to the parent mandate's cedar_action_set.
issued_at	ISO-8601 timestamp of issuance.
so_uuid	The SO UUID this mandate is bound to. Per INV-6 (draft-sato-soos-mjwt Section 4), a mandate is SO-instance-bound.

The mandate issuance tree is used to compute the CASCADE_TO_DESCENDANTS revocation scope defined in Section 3.5.

3.3. SO-Type-Bound Creation Mandates

INV-6 binds a mandate JWT to a specific SO UUID. This creates a bootstrapping dependency: creating an SO requires a mandate, but the SO UUID does not exist until creation.

SOOS resolves this with SO-Type-bound creation mandates. A creation mandate is scoped to an SO Type identifier, not to an SO instance UUID. It grants authority to call gec.createSovereignObject() for SOs of the specified type.

SO-Type-bound creation mandates MUST record:

creation_mandate	Boolean flag indicating this mandate authorises SO creation, not transitions.
so_type	The SO Type Registry identifier the mandate is scoped to.
so_type_version	The version constraint, if any.

The GEC MUST enforce that a creation mandate is only accepted at the gec.createSovereignObject() call, not at gec.transition(). An SO-instance-bound mandate MUST NOT be accepted at gec.createSovereignObject().

3.4. Creation Principal Classes

Every SO instance is created by exactly one of three Creation Principal Classes:

HUMAN_DIRECT
: A human operator creates the SO instance directly via an application. No parent mandate is required. The creating principal MUST hold a human-backed Ed25519 Party Registry key.

AGENT_DELEGATED
: An agent creates the SO instance under a mandate that explicitly includes SO creation authority for a given SO Type (Section 3.3). The creating agent MUST present a valid SO-Type-bound creation mandate at gec.createSovereignObject().

AGENT_AUTONOMOUS
: An agent with standing Party Registry creation rights for a specific SO Type creates the instance without a per-invocation mandate. Standing rights are declared in the Party Registry at agent registration time by a human principal.

The creation_principal_class MUST be recorded in the

CREATE_SOVEREIGN_OBJECT GEC event (Section 7).

Cedar evaluates against the SO Type's creation policy before creation occurs. The default result is PERMIT. The SO Type designer MAY declare DENY rules for: class restriction (e.g., AGENT_AUTONOMOUS prohibited for this type), rate control, operator suspension, or cross-SO dependency conditions.

3.5. Cross-Mandate Revocation Cascade

When a mandate revocation is issued with revocation_scope: CASCADE_TO_DESCENDANTS (as defined in draft-sato-soos-mjwt Section 5.3), the GEC MUST look up all mandate JWTs in the Party Registry whose issuance chain includes any revoked jti as an ancestor.

All descendant jti values MUST be added to the Revocation Registry atomically with the parent revocation. The cascade MUST be recorded in the single MANDATE_REVOCATION_ISSUED event -- not as separate per-descendant events. One human decision; one kernel action; complete audit trail.

This invariant ensures that revoking an orchestrator mandate terminates all specialist sub-agent mandates simultaneously, without requiring the revoking human to enumerate the delegation tree.

4. SO Instance Topology Types

4.1. Topology Classification

SOOS recognises five SO Instance Topology Types describing how multiple SO instances relate to each other at runtime. These topologies are not mutually exclusive within a complex application: a single workflow may exhibit Linear Chain structure at the top level while individual nodes contain Parallel Fan-Out sub-topologies.

The topology classification is architectural guidance for SO Type designers and orchestrator implementors. The kernel operates on individual SOs one transition at a time regardless of topology. INV-1 through INV-14 apply uniformly across all topology types.

4.2. Topology 1: Linear Chain (Sequential Pipeline)

A parent SO instance owns a defined sequence of child SO instances that complete in order. The parent state machine gates the creation of each subsequent child on the prior child reaching a terminal state. ProximityEvents (defined in draft-sato-soos-aep Section 7.3.2) deliver completion signals from child to parent.

Example: A travel booking workflow comprising FlightOut_SO -> Hotel_SO -> Activity_SO -> FlightReturn_SO. The parent BookingObject_SO enters a "legs pending" parallel state; each leg SO's terminal state fires a ProximityEvent to the parent.

Kernel requirement: The parent SO stores child SO UUIDs as Zone A cross-references. The ProximityEvent carries the child's so_uuid and terminal state as payload. The parent state machine uses these references to gate its own transitions.

This topology is fully supported by the current SOOS kernel.

4.3. Topology 2: Parallel Fan-Out (Concurrent Siblings)

Multiple child SO instances run simultaneously under a common

parent. The parent SO aggregates completion signals from children according to a declared aggregation rule: ALL_COMPLETE, ANY_COMPLETE, or QUORUM(n).

The SO Cluster Manager (Section 5) provides the coordination primitives for this topology. The orchestrator declares a cluster after creating the member SOs; the GEC evaluates the aggregation rule at each PARALLEL_FAN_OUT member transition and fires CLUSTER_AGGREGATION_CONDITION_MET when the condition is satisfied.

Example: A supplier availability check dispatching simultaneously to five vendor SO instances, proceeding when ANY_COMPLETE.

4.4. Topology 3: Directed Acyclic Graph (DAG)

Multiple child SO instances execute in parallel. Not all succeed. Terminated children (dead ends) are informational data points for surviving paths. The DAG shape is defined at SO Type design time.

This topology requires:

(a) Dynamic SO creation under AGENT_DELEGATED creation mandates.
(b) SO Cluster membership that can accommodate terminal members while the cluster remains active.
(c) Cross-SO Zone A data flow: surviving children reference the Zone A data produced by terminated siblings.

The GEC does not enforce the DAG structure. The orchestrator is responsible for declaring cluster membership and managing cross-SO references in Zone A.

Example: A multi-path experimental workflow where several hypothesis SOs are created simultaneously, results of terminated experiments feed surviving paths, and one path reaches the target.

4.5. Topology 4: Dynamic and Emergent

Child SO instances emerge at runtime based on execution outcomes. The topology shape is itself an outcome of execution, not a pre-defined structure. This is the most general case, of which Topology 3 (DAG) is a constrained special case.

This topology requires:

(a) Dynamic SO creation under AGENT_DELEGATED mandates at each step where the orchestrator determines a sub-task warrants governance.
(b) L1-16 DYNAMIC cluster membership (addClusterMember, removeClusterMember).
(c) Cedar evaluation at each dynamic creation step.

Example: An investigation workflow where an orchestrator SO spawns hypothesis SOs dynamically; each hypothesis spawns experiment SOs on positive results; experiments spawn retry SOs on partial failure. The topology is determined by data, not design.

4.6. Topology 5: Cyclic and Re-entrant

An SO returns to a prior state that it has already occupied. This is not a new child SO -- it is the same instance re-entering a state in its own state machine.

This topology is handled by the STATE_REVERSAL TransitionDeclaration mechanism defined in draft-sato-soos-sov Section 5. The SO Type designer declares which transitions are STATE_REVERSALs and which settled obligations they disturb. The authority to

execute a STATE_REVERSAL is determined by the settled_obligations_disturbed field.

Example: A quality review process where a batch SO can cycle between PROCESSING and QUALITY_REVIEW states multiple times before reaching APPROVED or REJECTED.

This topology does not require the SO Cluster Manager. It is resolved entirely by the state machine and TransitionDeclaration mechanism.

4.7. Kernel Effects by Topology

INV-1 through INV-14 do not change for any topology. The kernel operates on individual SOs one transition at a time regardless of the number of agents or SOs involved in the containing workflow.

What changes by topology is the orchestration layer above the kernel:

- Cross-SO Zone A references are the linking mechanism for all topology types. A parent SO stores child SO UUIDs in Zone A; this is how the topological relationship is represented in the governed record.
- The SO Cluster Manager (Section 5) provides the coordination layer for Topologies 2, 3, and 4.
- ProximityEvents (Topology 1) and CLUSTER_AGGREGATION_CONDITION_MET (Topologies 2, 3, 4) are the signals by which completion propagates upward in the topology.
- Topology 5 requires no additional kernel mechanism beyond STATE_REVERSAL TransitionDeclarations.

5. SO Cluster Coordination

5.1. The SO Granularity Rule

Before specifying cluster primitives, this section provides normative guidance on when a discrete SO instance is warranted, to prevent the performance and governance costs of over-granular SO creation in multi-agent workflows.

An SO instance is warranted when a thing requires at least one of:

(1) Governed state: the thing has discrete lifecycle phases requiring Cedar-enforced authority to transition.

(2) HEM eligibility: a human must be able to halt and decide at this unit’s boundary.

(3) Mandate scoping: different agents or humans require different authority over this thing independently.

(4) Audit accountability: an independent governance history of this specific unit is required.

Data that does not meet any of these criteria SHOULD be modelled as Zone B attachments on an existing SO, not as a separate SO instance.

This guidance is advisory, not normative. The SO Type designer determines the appropriate granularity for their domain.

5.2. Cluster Declaration Protocol

A cluster is declared after its member SOs are created. The cluster is a coordination and visibility overlay on existing SO instances; it does not itself hold Cedar-governed state.

Timing: Member SOs MUST be created individually before cluster declaration. The GEC does not support batch SO creation.

Authority: All three Creation Principal Classes (Section 3.4) may declare clusters. Cluster declaration is Cedar-evaluated against the DeclareCluster Cedar action (Section 8).

SO state impact: Cluster declaration does not alter the state of any member SO. The cluster is coordination overlay only.

Merge and split: Merging two clusters into one, and splitting one cluster into two, are first-class operations. Both require Cedar evaluation against MergeCluster and SplitCluster (Section 8). When clusters are merged, the orchestrator mandate governing the resulting cluster MUST satisfy INV-4 with respect to all mandates held by agents operating on any member SO.

Orchestrator declaration on merge or split: The orchestrator MUST explicitly declare the new cluster structure after a merge or split. The GEC does not infer cluster structure from agent behaviour.

Mandate scope on merge: INV-4 (Narrowing Property) is enforced at individual SO transition time, not at merge declaration time.

New GEC events: CLUSTER_DECLARED, CLUSTER_MERGED, CLUSTER_SPLIT (Section 7).

Conformance rules:

CONF-MAD-01: A cluster MUST NOT be declared with zero members.

CONF-MAD-02: A CLUSTER_MERGED event MUST record both source cluster IDs and the resulting cluster ID.

CONF-MAD-03: A CLUSTER_SPLIT event MUST record the source cluster ID and both resulting cluster IDs.

5.3. Cluster Membership Model

Each cluster is declared with a membership model at creation time. The membership model is immutable after declaration.

STATIC membership: Members are declared at cluster creation and cannot change. addClusterMember and removeClusterMember are not permitted on STATIC clusters.

DYNAMIC membership: Members may be added and removed via addClusterMember and removeClusterMember. These operations are Cedar-evaluated (Section 8).

Terminal state handling: When a cluster member SO reaches a terminal state, the GEC fires a CLUSTER_MEMBER_REACHED_TERMINAL ProximityEvent to all active sessions on the cluster. The orchestrator MUST explicitly declare member removal via removeClusterMember. The GEC does not automatically remove terminal members. This preserves INV-5 (every transition has exactly one Event Log entry) and makes terminal-state handling self-correcting under orchestrator failure.

Completed member retention: Terminal members are retained in the Cluster Registry with TERMINAL status until CLUSTER_DISSOLVED.

This allows the orchestrator to query the full cluster history at any point in the workflow.

New GEC events: CLUSTER_MEMBER_ADDED, CLUSTER_MEMBER_REMOVED (Section 7).

New ProximityEvent: CLUSTER_MEMBER_REACHED_TERMINAL (Section 7).

Conformance rules:

CONF-MAD-04: addClusterMember MUST be rejected on STATIC clusters.

CONF-MAD-05: removeClusterMember MUST be rejected for members in non-TERMINAL states on STATIC clusters.

CONF-MAD-06: CLUSTER_MEMBER_REACHED_TERMINAL MUST be delivered to all active sessions before any subsequent SENSE delivery on the affected cluster.

CONF-MAD-07: The orchestrator MUST explicitly call removeClusterMember before dissolveCluster if any members remain in non-TERMINAL states.

5.4. Cluster Registry

The GEC MUST maintain a Cluster Registry: an in-memory index of all declared clusters, their members, current member states, and aggregation status.

INV-14 (Cluster Registry Reconstruction): The Cluster Registry MUST be rebuilt from the Event Log on GEC restart. It is a performance projection of the Event Log, not a second source of truth. The pattern is identical to the Revocation Registry defined in draft-sato-soos-mjwt Section 5.2.

The Cluster Registry MUST maintain, for each cluster:

cluster_id	UUID v4, GEC-assigned at declaration.
membership_model	STATIC or DYNAMIC.
members[]	For each member SO: so_uuid, current_so_state (projection updated on every member STATE_TRANSITION), lifecycle_phase, membership_status (ACTIVE or TERMINAL).
aggregation_rule	Declared aggregation condition, if any.
aggregation_fired	Boolean; TRUE after the aggregation condition has been satisfied once.
orchestrator_session	The session_id of the orchestrator agent.

Performance contract:

- getClusterStatus(): O(1), target <1ms.
- Cluster Registry update on member SO transition: O(1).
- INV-14 Event Log replay: O(e) where e is the number of cluster-related events. Accepted, same class as Revocation Registry.

CONF-MAD-08: The Cluster Registry MUST be rebuilt from the Event Log on GEC restart before processing any cluster queries.

5.5. Cluster-Enriched Cedar Evaluation

The GEC MUST NOT evaluate Cedar at the cluster scope. Cedar evaluates at the individual SO transition scope only. This preserves the kernel contract (draft-sato-soos-aep Section 4.1): every transition is atomic, Cedar-evaluated, and Event-Log-committed at the individual SO level.

However, individual SO transitions on cluster members SHOULD carry cluster context to enable cluster-aware Cedar policies. The kernel MUST inject a cluster_context attribute into the Cedar evaluation context for every transition on a cluster member SO. The cluster_context attribute is populated from the Cluster Registry (O(1) lookup) and carries:

cluster_id	The cluster this SO belongs to.
cluster_size	Total member count.
terminal_count	Count of members in TERMINAL state.
aggregation_status	PENDING or CONDITION_MET.
is_last_active	Boolean; TRUE if this is the last non-terminal member.

This enables Cedar policies such as:
"PERMIT this action only if aggregation_status is CONDITION_MET"
or "REQUIRE HEM if is_last_active is TRUE and action is ROLLBACK".

CONF-MAD-09: cluster_context MUST be populated from the Cluster Registry before Cedar evaluation on any cluster member transition.

5.6. SO Cluster Manager (L1-16)

L1-16 is a normative SOOS OS service. Implementations conforming to this document MUST provide the SO Cluster Manager with the following six functions:

declareCluster(member_so_uuids[], membership_model,
aggregation_rule, orchestrator_session_id)
Declares a new cluster. Cedar evaluation: DeclareCluster.
Returns: cluster_id (UUID v4, GEC-assigned).
Fires: CLUSTER_DECLARED GEC event.

getClusterStatus(cluster_id)
Pure read. Returns the Cluster Registry entry for the cluster.
No Cedar evaluation. O(1), no Event Log scan.
Returns: cluster_id, membership_model, members[],
aggregation_status, aggregation_fired.

addClusterMember(cluster_id, so_uuid)
Adds a member to a DYNAMIC cluster. Cedar: AddClusterMember.
Fires: CLUSTER_MEMBER_ADDED GEC event.

removeClusterMember(cluster_id, so_uuid)
Removes a member from a cluster. Cedar: RemoveClusterMember.
MUST be rejected if so_uuid is in non-TERMINAL state on a
STATIC cluster (CONF-MAD-05).
Fires: CLUSTER_MEMBER_REMOVED GEC event.

mergeCluster(cluster_id_a, cluster_id_b)
Merges two clusters into one. Cedar: MergeCluster.
Fires: CLUSTER_MERGED GEC event. Both source clusters are
dissolved; a new cluster_id is assigned.

splitCluster(cluster_id, member_set_a[], member_set_b[])
Splits one cluster into two. Cedar: SplitCluster.
Fires: CLUSTER_SPLIT GEC event. The source cluster is
dissolved; two new cluster_ids are assigned.

dissolveCluster(cluster_id)

Dissolves the cluster. Cedar: DissolveCluster.
MUST be rejected if any members are in non-TERMINAL state
(CONF-MAD-10).
Fires: CLUSTER_DISSOLVED GEC event.

CONF-MAD-10: dissolveCluster MUST be REJECTED if any cluster member SO is in non-TERMINAL state.

5.7. Aggregation Rules

The orchestrator MAY declare an aggregation rule at cluster creation time. The aggregation rule defines a condition on cluster member states that, when satisfied, triggers the CLUSTER_AGGREGATION_CONDITION_MET ProximityEvent.

Supported aggregation conditions:

ALL_COMPLETE	All member SOs have reached terminal states.
ANY_COMPLETE	At least one member SO has reached a terminal state.
QUORUM(n)	At least n member SOs have reached terminal states, where n is declared at creation time and MUST be between 1 and the total member count.

The GEC evaluates the aggregation rule at every PARALLEL_FAN_OUT member SO transition using the Cluster Registry projection (no Event Log scan required).

When the aggregation condition is first satisfied and aggregation_fired is FALSE:

(a) The kernel fires CLUSTER_AGGREGATION_CONDITION_MET as a ProximityEvent to the orchestrator session.

(b) The kernel sets aggregation_fired to TRUE in the Cluster Registry. The condition fires exactly once.

Session availability: If no active orchestrator session exists when the condition is satisfied, the ProximityEvent is queued and delivered at the start of the next orchestrator session. This follows the existing ProximityEvent delivery model.

Conformance rules:

CONF-MAD-11: Aggregation rule evaluation MUST use the Cluster Registry projection, not an Event Log scan.

CONF-MAD-12: CLUSTER_AGGREGATION_CONDITION_MET MUST be delivered exactly once per cluster lifetime. Subsequent member transitions after aggregation_fired is TRUE MUST NOT re-fire the event.

CONF-MAD-13: QUORUM(n) where n exceeds the member count MUST be rejected at cluster declaration time.

5.8. Visibility Extensions

Multi-agent visibility is a first-class concern. This section specifies extensions to three existing OS service surfaces that enable cluster-aware visibility.

5.8.1. Agent Visibility: L1-01 (Live Permission Map)

L1-01 is extended to support cluster-scope permission queries. The new function getClusterPermittedActions(cluster_id) returns,

for each member SO, the set of Cedar-permitted actions under the calling agent's current mandate. This enables an orchestrator to determine the available action space across all cluster members in a single call, using the Cluster Registry projection for member enumeration and individual Cedar evaluation per member.

Performance contract: O(1) per member via Cluster Registry; Cedar evaluation is O(m) where m is the member count.

5.8.2. Agent Visibility: L1-09 (Transition Graph)

L1-09 is extended to support cluster-scope path planning. The new function getClusterTransitionGraph(cluster_id) returns a cluster-scope path plan with a precondition field orchestrator_path_precondition: AGGREGATION_CONDITION_MET.

This allows the orchestrator to declare its intended path conditional on aggregate completion, enabling plan validation before any individual member transition is executed.

5.8.3. Human Visibility: HEMContext

The HEMContext schema defined in draft-sato-soos-hem Section 7.8.3 is extended with a cluster_context field. When a HEM is invoked on a cluster member SO, the HEMContext delivered to the deciding human principal MUST include:

cluster_id	The cluster this SO belongs to.
cluster_summary	Member list with current states.
aggregation_status	PENDING or CONDITION_MET.
is_triggering_so	TRUE for the member SO that triggered HEM.

This enables the human reviewer to understand the multi-agent workflow state at the time of escalation, not only the individual SO state.

CONF-MAD-14: cluster_context in HEMContext MUST be populated from the Cluster Registry at HEM_INVOKED time.

6. Orchestrator-Specialist Model

6.1. GEE Orchestration Mode

The Goal Execution Engine (GEE) orchestration mode, defined in draft-sato-soos-aep Section 7.6, is the normative operating model for orchestrator agents in SOOS. In GEE mode, the GEE inverts control: rather than the orchestrator agent driving its own AEP loop, the GEE calls the orchestrator's reason() function as a service within a GEC-driven loop.

The orchestrator exposes a single reasoning interface:

```
reason(context_package: ContextPackage) -> ReasoningOutput
```

The GEE is responsible for:

(a) Delivering normative Context Packages to the orchestrator.
(b) Constructing the IDP from the orchestrator's ReasoningOutput.
(c) Calling gec.transition() and kernel cluster operations on behalf of the orchestrator.
(d) Issuing sub-agent mandates from the orchestrator's mandate.

The orchestrator MUST NOT call gec.transition() or cluster operations directly in GEE mode (CONF-GEE-06 of draft-sato-soos-aep).

6.2. Orchestrator Mandate Scope

An orchestrator agent operates under a mandate issued by a human principal. The orchestrator mandate defines the Cedar action set from which all sub-agent mandates must be derived (INV-4).

The orchestrator mandate MUST be SO-instance-bound (INV-6) for each SO the orchestrator will transition directly. For SOs the orchestrator creates and then delegates to specialists, the orchestrator holds a SO-Type-bound creation mandate (Section 3.3) plus the Cedar actions required to declare and manage the cluster.

For cluster-spanning workflows, the orchestrator mandate MUST include all Cedar cluster management actions required for the intended topology: DeclareCluster, AddClusterMember (for DYNAMIC clusters), MergeCluster or SplitCluster (if topology evolution is expected), and DissolveCluster.

6.3. Specialist Agent Mandate Issuance

The orchestrator issues mandates to specialist agents. Each specialist mandate MUST satisfy INV-4: the specialist's Cedar action set MUST be a strict subset of the orchestrator's Cedar action set on the same SO.

Each specialist mandate MUST be SO-instance-bound (INV-6) to a specific member SO UUID. An orchestrator MUST NOT issue a mandate that grants a specialist authority over multiple SO instances in a single mandate. If a specialist requires authority over multiple SOs, the orchestrator MUST issue separate mandates, one per SO, each individually INV-4-compliant.

The mandate issuance tree (Section 3.2) records each issuance relationship, enabling full provenance reconstruction at audit time.

6.4. Sub-Agent Failure Recovery

When a specialist agent fails, expires, or is revoked, the orchestrator must recover the cluster state. The primary recovery signal is CLUSTER_MEMBER_REACHED_TERMINAL ProximityEvent.

The recovery protocol is:

(1) The orchestrator receives CLUSTER_MEMBER_REACHED_TERMINAL for the failed specialist's member SO.

(2) The orchestrator calls getClusterStatus() to determine the full cluster state.

(3) Based on the cluster state and the member SO's terminal state, the orchestrator determines whether to:
   (a) Continue -- the aggregation rule tolerates this failure (e.g., QUORUM(n) is still achievable).
   (b) Invoke HEM -- the failure is outside the orchestrator's confident decision scope.
   (c) Dissolve -- the workflow cannot continue; the orchestrator initiates compensating actions on all remaining members.

If the sub-agent's mandate was revoked (not expired), the CASCADE_TO_DESCENDANTS behaviour (Section 3.5) ensures all further transitions under the revoked mandate are blocked.

7. Kernel Events

This section specifies the GEC events introduced by this document. All GEC events MUST be signed by the KIA keypair

(INV-9 of draft-sato-soos-kia).

7.1. CREATE_SOVEREIGN_OBJECT

```
CREATE_SOVEREIGN_OBJECT {
  event_type:               "CREATE_SOVEREIGN_OBJECT",
  event_id:                 <UUID>,
  timestamp:                <ISO-8601 microsecond>,
  so_uuid:                  <UUID v4, GEC-assigned>,
  so_type:                  <type_registry_id>,
  so_type_version:          <semver>,
  creation_principal_class: "HUMAN_DIRECT" |
                            "AGENT_DELEGATED" |
                            "AGENT_AUTONOMOUS",
  creation_mandate_jti:     <jti> | null,
  initial_state:            <state_name>,
  initial_zone_a_data:      { <typed_graph_nodes> },
  cedar_creation_result:    "PERMIT" | "DENY",
  gec_signature:         <Ed25519 over canonical JSON>
}
```

creation_mandate_jti is null for HUMAN_DIRECT and AGENT_AUTONOMOUS creation. MUST be non-null for AGENT_DELEGATED creation.

7.2. CLUSTER_DECLARED

```
CLUSTER_DECLARED {
  event_type:               "CLUSTER_DECLARED",
  event_id:                 <UUID>,
  timestamp:                <ISO-8601 microsecond>,
  cluster_id:               <UUID v4, GEC-assigned>,
  membership_model:         "STATIC" | "DYNAMIC",
  member_so_uuids:          [ <UUID>, ... ],
  aggregation_rule:         "ALL_COMPLETE" | "ANY_COMPLETE" |
                            "QUORUM" | null,
  aggregation_quorum_n:     <integer> | null,
  orchestrator_session_id:  <UUID>,
  orchestrator_mandate_jti: <jti>,
  cedar_result:             "PERMIT",
  gec_signature:         <Ed25519 over canonical JSON>
}
```

7.3. CLUSTER_MEMBER_ADDED

```
CLUSTER_MEMBER_ADDED {
  event_type:          "CLUSTER_MEMBER_ADDED",
  event_id:            <UUID>,
  timestamp:           <ISO-8601 microsecond>,
  cluster_id:          <UUID>,
  so_uuid:             <UUID>,
  requesting_agent_id: <party_registry_id>,
  mandate_jti:         <jti>,
  cedar_result:        "PERMIT",
  gec_signature:    <Ed25519 over canonical JSON>
}
```

7.4. CLUSTER_MEMBER_REMOVED

```
CLUSTER_MEMBER_REMOVED {
  event_type:          "CLUSTER_MEMBER_REMOVED",
  event_id:            <UUID>,
  timestamp:           <ISO-8601 microsecond>,
  cluster_id:          <UUID>,
  so_uuid:             <UUID>,
  member_final_state:  <state_name>,
```

```
  requesting_agent_id: <party_registry_id>,
  mandate_jti:         <jti>,
  cedar_result:        "PERMIT",
  gec_signature:    <Ed25519 over canonical JSON>
}
```

7.5. CLUSTER_MERGED

```
CLUSTER_MERGED {
  event_type:               "CLUSTER_MERGED",
  event_id:                 <UUID>,
  timestamp:                <ISO-8601 microsecond>,
  source_cluster_id_a:      <UUID>,
  source_cluster_id_b:      <UUID>,
  resulting_cluster_id:     <UUID v4, GEC-assigned>,
  membership_model:         "STATIC" | "DYNAMIC",
  requesting_agent_id:      <party_registry_id>,
  cedar_result:             "PERMIT",
  gec_signature:         <Ed25519 over canonical JSON>
}
```

7.6. CLUSTER_SPLIT

```
CLUSTER_SPLIT {
  event_type:               "CLUSTER_SPLIT",
  event_id:                 <UUID>,
  timestamp:                <ISO-8601 microsecond>,
  source_cluster_id:        <UUID>,
  resulting_cluster_id_a:   <UUID v4, GEC-assigned>,
  member_uuids_a:           [ <UUID>, ... ],
  resulting_cluster_id_b:   <UUID v4, GEC-assigned>,
  member_uuids_b:           [ <UUID>, ... ],
  requesting_agent_id:      <party_registry_id>,
  cedar_result:             "PERMIT",
  gec_signature:         <Ed25519 over canonical JSON>
}
```

7.7. CLUSTER_DISSOLVED

```
CLUSTER_DISSOLVED {
  event_type:               "CLUSTER_DISSOLVED",
  event_id:                 <UUID>,
  timestamp:                <ISO-8601 microsecond>,
  cluster_id:               <UUID>,
  final_member_states:      [ { so_uuid, final_state }, ... ],
  requesting_agent_id:      <party_registry_id>,
  cedar_result:             "PERMIT",
  gec_signature:         <Ed25519 over canonical JSON>
}
```

7.8. ProximityEvent: CLUSTER_MEMBER_REACHED_TERMINAL

This ProximityEvent is fired when a cluster member SO reaches a terminal state. It is delivered to the orchestrator session per the ProximityEvent delivery semantics of draft-sato-soos-aep Section 7.3.2.

```
ProximityEvent {
  condition_id:    <cluster_id>:<so_uuid>,
  condition_type:  "CLUSTER_MEMBER_REACHED_TERMINAL",
  current_value:   <terminal_state_name>,
  threshold_value: "TERMINAL",
  proximity_pct:   1.0,
  cluster_id:      <UUID>,
  so_uuid:         <UUID>,
  remaining_active_count: <integer>
```

```
}
```

7.9. ProximityEvent: CLUSTER_AGGREGATION_CONDITION_MET

This ProximityEvent is fired when the cluster's declared aggregation condition is first satisfied.

```
ProximityEvent {
  condition_id:      <cluster_id>:AGGREGATION,
  condition_type:    "CLUSTER_AGGREGATION_CONDITION_MET",
  current_value:     <satisfied_count>,
  threshold_value:   <required_count>,
  proximity_pct:     1.0,
  cluster_id:        <UUID>,
  aggregation_rule:  "ALL_COMPLETE" | "ANY_COMPLETE" | "QUORUM",
  satisfied_members: [ <so_uuid>, ... ]
}
```

8. Cedar Actions

The following Cedar actions are introduced by this document. All actions MUST be registered in the SOOS Cedar namespace.

SOOS::Action::CreateSovereignObject
: Required to call gec.createSovereignObject(). Cedar evaluates against the SO Type's creation policy.

SOOS::Action::DeclareCluster
: Required to call L1-16 declareCluster().

SOOS::Action::AddClusterMember
: Required to call L1-16 addClusterMember().

SOOS::Action::RemoveClusterMember
: Required to call L1-16 removeClusterMember().

SOOS::Action::MergeCluster
: Required to call L1-16 mergeCluster().

SOOS::Action::SplitCluster
: Required to call L1-16 splitCluster().

SOOS::Action::DissolveCluster
: Required to call L1-16 dissolveCluster().

9. Conformance

A conforming SOOS multi-agent implementation MUST satisfy all of the following requirements. Items marked REJECT cause the kernel to reject the triggering operation.

CONF-MAD-01 A cluster MUST NOT be declared with zero members. (REJECT)

CONF-MAD-02 A CLUSTER_MERGED event MUST record both source cluster IDs and the resulting cluster ID. (REJECT)

CONF-MAD-03 A CLUSTER_SPLIT event MUST record the source cluster ID and both resulting cluster IDs. (REJECT)

CONF-MAD-04 addClusterMember MUST be REJECTED on STATIC clusters.

CONF-MAD-05 removeClusterMember MUST be REJECTED for members in non-TERMINAL states on STATIC clusters.

CONF-MAD-06 CLUSTER_MEMBER_REACHED_TERMINAL MUST be delivered to all active sessions on the cluster before any subsequent SENSE delivery on the affected cluster.

CONF-MAD-07 The orchestrator MUST explicitly call removeClusterMember before dissolveCluster if any members remain in non-TERMINAL states. (REJECT if non-TERMINAL members exist)

CONF-MAD-08 The Cluster Registry MUST be rebuilt from the Event Log on GEC restart before processing any cluster queries. (REJECT cluster queries before rebuild is complete)

CONF-MAD-09 cluster_context MUST be populated from the Cluster Registry before Cedar evaluation on any cluster member transition.

CONF-MAD-10 dissolveCluster MUST be REJECTED if any cluster member SO is in non-TERMINAL state.

CONF-MAD-11 Aggregation rule evaluation MUST use the Cluster Registry projection, not an Event Log scan.

CONF-MAD-12 CLUSTER_AGGREGATION_CONDITION_MET MUST be delivered exactly once per cluster lifetime.

CONF-MAD-13 QUORUM(n) where n exceeds the declared member count MUST be REJECTED at cluster declaration time.

CONF-MAD-14 cluster_context in HEMContext MUST be populated from the Cluster Registry at HEM_INVOKED time.

CONF-MAD-GEC-01: A conforming GEC implementation MUST enforce the Narrowing Property (INV-4) at mandate issuance time in the Party Registry. Enforcement solely at gec.transition() evaluation time does not satisfy this requirement.

CONF-MAD-GEC-02: A conforming GEC implementation MUST rebuild the Cluster Registry from the Event Log on GEC restart before processing any cluster queries. A GEC serving cluster queries from an unverified in-memory state without rebuild verification does not satisfy this requirement.

10. Open Issues

The following open questions are recorded for resolution in successor documents.

10.1. OQ-S-41: Cross-Principal SO Coordination

When two SOs from separate Party Registry principals (or separate SOOS instances) represent the same real-world transaction -- for example, an operator’s GuestStaySO and a traveler’s TravelBookingSO -- no normative coordination protocol exists for:

(a) Declaring the cross-principal relationship between the two SOs.
(b) Propagating state change signals between principals without violating each principal’s SO autonomy.
(c) Handling disputes where the two SOs record conflicting versions of events for the same real-world transaction.
(d) Coordinating Zone A data across principals without violating Zone A data residency requirements.

This is a genuine architectural gap. OQ-S-41 is a prerequisite

for cross-domain ATP deployment and for any multi-operator SOOS federation scenario. It constrains: Zone B cross-principal data sharing, GDPR cross-member accountability, and kernel multi-tenancy.

OQ-S-41 is tracked as HIGH priority. It does not block v1 single-operator deployments.

10.2. OQ-S-43: Nested Clusters

Whether a cluster of clusters (a cluster whose members are themselves cluster identifiers rather than SO instance identifiers) is a supported construct is unresolved. Nested clusters would support hierarchical multi-agent workflows with independent sub-workflow aggregation conditions. This is tracked as LOW priority and does not block v1.

10.3. OQ-S-44: Cluster-Scope Cedar Evaluation

Whether Cedar evaluation at the cluster scope (evaluating a Cedar policy that directly references cluster-level state rather than individual SO state) is warranted in a successor document is unresolved. The cluster_context Cedar attribute (Section 5.5) provides a pragmatic interim mechanism. OQ-S-44 is a prerequisite for OQ-S-43. Tracked as LOW priority; does not block v1.

11. Security Considerations

11.1. Confused Deputy Attack at Delegation Hops

The Narrowing Property (INV-4) structurally prevents authority amplification at each delegation hop. However, implementors MUST enforce INV-4 at mandate issuance time in the Party Registry, not only at evaluation time. An implementation that issues non-INV-4-compliant mandates and relies on Cedar evaluation alone to prevent authority amplification is non-conforming.

11.2. Cluster Scope as an Attack Surface

Cluster declarations and membership changes are Cedar-evaluated. However, the cluster_context Cedar attribute injected per-transition (Section 5.5) creates a new input surface for Cedar policy evaluation. Implementors MUST ensure that cluster_context is populated exclusively from the GEC-maintained Cluster Registry and cannot be supplied or manipulated by agents. An agent that can inject cluster_context values can potentially bypass Cedar policies conditioned on cluster state.

11.3. Ghost Execution on Cluster Dissolution

If an orchestrator mandate expires or is revoked while cluster members remain in non-TERMINAL states, the cluster may enter a state where no active agent has authority to dissolve it. The kernel MUST NOT automatically dissolve clusters. Operator intervention via a human-held Party Registry principal is required to dissolve clusters abandoned by their orchestrator.

Implementations SHOULD provide an operator-accessible kernel.forceDissolveCluster() function requiring human-principal authority, for use in recovery scenarios. This function is out of scope for v1 and deferred to a successor document.

11.4. Cascade Revocation in Large Delegation Trees

In workflows with deep delegation trees (orchestrator issuing to specialists who issue to sub-specialists), a CASCADE_TO_DESCENDANTS revocation may affect a large number of active sessions

simultaneously. Implementations MUST handle the atomic revocation of large descendant sets without partial failure. The MANDATE_REVOCATION_ISSUED event records the complete set of revoked jtis; the GEC MUST add all listed jtis to the Revocation Registry atomically.

12. IANA Considerations

This document has no IANA actions at this time. A successor document will define the SOOS Cedar action namespace registry.

13. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/rfc/rfc7519>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[SOOS-AEP] Sato, T., "Agent Execution Protocol", draft-sato-soos-aep-00, Work in Progress, May 2026.

[SOOS-HEM] Sato, T., "Human Escalation Mechanism", draft-sato-soos-hem-01, Work in Progress, May 2026.

[SOOS-KIA] Sato, T., "Kernel Identity and Attestation", draft-sato-soos-kia-00, Work in Progress, May 2026.

[SOOS-MJWT] Sato, T., "Mandate JWT", draft-sato-soos-mjwt-00, May 2026.

[SOOS-SOV] Sato, T., "Sovereign Object", draft-sato-soos-sov-00, May 2026.

14. Informative References

[SOOS-IDP] Sato, T., "Intent Declaration Primitive", draft-sato-soos-idp-03, May 2026.

[SPICE-ACTOR-CHAIN] Looker, T. et al., "SPICE Actor Chain", draft-mw-spice-actor-chain-05, Work in Progress, 2026.

[OAUTH-ATTENUATING] Niyikiza, J. et al., "Attenuating Agent Tokens", draft-niyikiza-oauth-attenuating-agent-tokens-00, Work in Progress, 2026.

[ICON-PS] Nair, et al., "Observability, Intervention and Control of Network Management Agents -- Problem Statement", Work in Progress, Internet-Draft, draft-nair-icon-problem-statement, 2026 (not yet submitted).

[AUDIT-BOF] Kuehlewind, M. and Birkholz, H., "Agent Use of Delegation and Interaction Traceability (AUDIT)", Work in Progress, Internet-Draft,

draft-kuehlewind-audit-architecture-00, May 2026.

[AUTHZEN] McGuinness, K. et al., "AuthZEN 1.0", OpenID Foundation, January 2026.

[OAUTH-TXN-TOKENS] Tulshibagwale, A., Fletcher, G., and P. Kasselman, "Transaction Tokens", draft-ietf-oauth-transaction-tokens-08, Work in Progress, 2026.

Appendix B. Related Work

This appendix situates MAD within the IETF multi-agent landscape. The central observation is that the agentic AI standards stack operates across three distinct layers, each addressed by different working groups and individual submissions:

- The credential/token layer: how authority is represented and presented (SPICE, OAuth, WIMSE).
- The communication/session layer: how agents exchange messages and manage sessions (ACP, A2A).
- The governance-state layer: what the enforcement environment guarantees happens to governed objects when an agent acts.

MAD operates exclusively at the governance-state layer. No current IETF working group charter claims this layer as primary scope. The capability coverage matrix published in the IETF WG Charter Analysis (ACP / WIMSE / OAuth / WebBotAuth / AIPREF / DAWN, v0.01, 2026) confirms that policy enforcement is out-of-scope or partial for all six active and proposed agentic AI working groups.

B.1. SPICE Actor Chain

draft-mw-spice-actor-chain-05 [SPICE-ACTOR-CHAIN] defines a mechanism for preserving the delegation provenance of an agent action: which principals authorised which hops, in what sequence, with what disclosure profile (full chain, subset, or committed).

MAD and SPICE actor-chain solve adjacent but distinct problems. SPICE answers: who was in the delegation chain and can you prove it? INV-4 answers: what was each hop in the chain permitted to cause? SPICE is attribution; INV-4 is constraint. Both are necessary for a complete multi-agent security architecture.

A conforming SOOS orchestrator SHOULD include the mandate JWT fingerprint (SHA-256 of the mandate JWT) in the SPICE actor-chain for each delegation hop. This makes the SOOS mandate issuance tree (Section 3.2) directly traceable in cross-system audit contexts without requiring the verifying party to have access to the SOOS Party Registry.

The delegation_chain JWT claim used in draft-sato-soos-idp-03 and draft-sato-soos-hem-01 follows the SPICE actor-chain profile and should be read in conjunction with this appendix.

B.2. OAuth Attenuating Agent Tokens

draft-niyikiza-oauth-attenuating-agent-tokens-00 [OAUTH-ATTENUATING] defines a mechanism for issuing attenuated tokens to sub-agents without requiring an Authorization Server round-trip at each delegation hop. The attenuated token carries a narrowed scope set derived from the parent token.

This mechanism addresses the same authority-narrowing requirement

as INV-4 but at the OAuth scope layer rather than the Cedar action layer. OAuth scopes are coarser than Cedar actions for the agentic context: a scope such as "booking:write" does not distinguish between confirming a booking and cancelling one, both of which a Cedar policy can express as distinct actions with distinct authority requirements.

The two mechanisms are complementary. An agent system MAY use OAuth attenuating tokens at the API authentication layer while using SOOS mandate JWTs and INV-4 at the governance-state layer. The attenuating token gates API access; the mandate JWT and Cedar policy gate state transitions on Sovereign Objects.

B.3. OAuth Transaction Tokens

draft-ietf-oauth-transaction-tokens-08 [OAUTH-TXN-TOKENS] (Tulshibagwale, Fletcher, Kasselman) defines Transaction Tokens (Txn-Tokens): short-lived, signed JWTs that propagate user identity, workload identity, and authorisation context through the call graph within a trusted domain.

Txn-Tokens address call graph identity propagation: ensuring that every service in a multi-service call chain knows who initiated the request and what authorisation context applies. MAD mandates address what state transitions each agent in that call chain is permitted to cause on governed objects.

A SOOS agent SHOULD embed its mandate JWT fingerprint in the request_context field of a Txn-Token when making calls to external services. This gives the receiving service both the Txn-Token’s identity chain and the Cedar action set the SOOS agent is operating under, enabling the receiving system to apply its own policy against the SOOS authority context without implementing the full SOOS kernel.

draft-araut-oauth-transaction-tokens-for-agents-06 extends Txn-Tokens specifically for agent-based workloads using the act field for agent identity. This extension is directly compatible with the SOOS mandate JWT as the act claim source.

B.4. Agent Communication Protocol (ACP)

The Agent Communication Protocol (Proposed WG / BoF at IETF 126) defines the communication and session layer for agent-to-agent and agent-to-tool interaction: message formats, session management, capability negotiation, and long-lived multi-modal sessions.

ACP has formally agreed to defer authentication to WIMSE and authorisation to OAuth. MAD occupies the governance-state layer that ACP’s authorisation deferral leaves open. When an ACP message arrives at an agent, MAD governs what state transitions the receiving agent is permitted to cause on Sovereign Objects as a result of acting on that message.

The relationship is compositional without overlap: ACP is the channel; MAD is the governance of what the channel causes in the state of governed objects. ACP’s per-operation user confirmation requirement (primary scope in the ACP capability matrix) maps directly to SOOS HEM Class 1 (HEM_MANDATORY) trigger behaviour.

B.5. A2A Protocol

The Agent-to-Agent Protocol (Google) defines agent-to-agent coordination via Agent Cards, capability negotiation, and task hand-off. Agent Cards are self-asserted capability declarations with no cryptographic verifier in the loop in the base

specification.

The SO Type Registry defined in draft-sato-soos-sov provides the operator-side equivalent of discoverable capabilities: SO Type definitions are versioned, signed, and registry-governed. A SOOS SO Type Registry entry SHOULD be exportable as an A2A-compatible signed Agent Card, providing verifiable capability advertisement for the SOOS agent population.

MAD's SO Cluster Manager (Section 5.6) provides governance primitives for the kind of multi-agent coordination A2A's task hand-off model requires, with the addition of kernel-enforced authority constraints and an append-only audit trail at each coordination step.

B.6. AuthZEN 1.0

AuthZEN 1.0 (OpenID Foundation, January 2026) [AUTHZEN] defines the Policy Enforcement Point / Policy Decision Point (PEP/PDP) separation model: a standardised API for external policy decisions independent of the agent process.

SOOS Cedar evaluation at the kernel layer is PEP/PDP separation enforced at the physics of the OS, not middleware: Cedar executes before XState on every transition (INV-3), and the evaluation is kernel-owned, not agent-owned. The AuthZEN model is the correct reference for how SOOS's Cedar enforcement layer relates to external policy infrastructure.

An implementation MAY expose the SOOS Cedar evaluation result via an AuthZEN-compatible API surface for consumption by external systems that need to reason about SOOS-governed agent authority without implementing the full SOOS kernel.

B.7. SOOS Companion Drafts

This document is one of nine SOOS IETF individual submissions. The full stack is:

draft-sato-soos-idp-03	Intent Declaration Primitive. Per-transition agent reasoning record. The IDP is the audit artifact that MAD's mandate and cluster events accompany.
draft-sato-soos-hem-01	Human Escalation Mechanism. Integrates with MAD via HEMContext cluster_context extension (Section 5.8.3).
draft-sato-soos-gar-01	Governance Audit Record. The SCITT-integrated audit layer for all events defined in Section 7 of this document.
draft-sato-soos-cap-00	Constitutional AI Protocol. Tier 0 and Tier 1 prohibitions enforced before Cedar. Applies to all agents in all topologies defined in Section 4.
draft-sato-soos-sov-00	Sovereign Object. Defines the SO lifecycle model, five-phase structure, and Zone A/B boundary that all cluster members in Section 5 are instances of.
draft-sato-soos-mjwt-00	Mandate JWT. Defines the credential format that INV-4

(Section 3.1) and the mandate issuance tree (Section 3.2) operate on.

draft-sato-soos-aep-00 — Agent Execution Protocol. Defines the per-agent SENSE/REASON/PLAN/ACT/OBSERVE loop within which MAD cluster operations occur.

draft-sato-soos-kia-00 — Kernel Identity and Attestation. Defines the keypair and signing authority for all GEC events in Section 7.

draft-sato-soos-pt-00 — Progressive Trust. Provides per-agent behavioural trust scores used in HEMContext to inform human decisions about multi-agent workflow escalations.

draft-sato-soos-faip-00 — Federated Agent Intelligence Protocol. Aggregates behavioural intelligence from MAD cluster execution records across operators, providing ecosystem-level insight into multi-agent workflow patterns.

B.8. ICON Initiative: Control Pillar

The ICON initiative [ICON-PS] defines Control as one of three pillars for autonomous agent governance in network management contexts, alongside Observability and Intervention.

Relationship: Direct composition. MAD addresses the ICON Control pillar in full. The two properties ICON requires of a Control mechanism are precisely what MAD specifies: authority delegation governance (INV-4 ensures control cannot amplify across delegation hops) and revocation propagation (CASCADE_TO_DESCENDANTS ensures a single revocation decision reaches all agents in the delegation tree immediately -- the stop-with-one-decision requirement).

In network management: when a network management orchestrator delegates routing decisions to segment specialists, MAD ensures (a) each specialist is authorized only for its specific segment; (b) if the orchestrator's authorization is revoked, all specialist sub-agents immediately lose authority to make further changes.

The SO Cluster Coordination Primitives (Section 5) address the ICON operational topology: network management parallel fan-out workflows with QUORUM or ALL_COMPLETE aggregation rules map directly to Topology 2 (Parallel Fan-Out) SO Clusters.

The ICON Intervention pillar is addressed by HEM; the Observability pillar by PT. Together HEM + PT + MAD cover all three ICON pillars.

Integration: ICON-conforming Control pillar implementations SHOULD adopt MAD's Narrowing Property and CASCADE_TO_DESCENDANTS as normative specifications for delegation authority governance and cascade revocation.

B.9. AUDIT Working Group

The AUDIT working group [AUDIT-BOF] is developing interoperable mechanisms for auditing AI agents across Internet protocols.

Relationship: Composition. MAD generates multi-agent audit events that AUDIT WG formats must accommodate: CREATE_SOVEREIGN_OBJECT, CLUSTER_DECLARED, CLUSTER_MEMBER_ADDED, CLUSTER_MEMBER_REMOVED,

CLUSTER_MERGED, and CLUSTER_DISSOLVED are candidate AUDIT WG interaction record entry types. The mandate issuance tree (Section 3.2) is the delegation provenance record that AUDIT WG Deliverable 2 must support.

Integration: AUDIT WG audit data models SHOULD include a multi-agent workflow record type carrying cluster_id, orchestrator_mandate_jti, member_so_uuids, and aggregation_rule_applied from the CLUSTER_DECLARED event schema.

B.10. DAWN Working Group

The DAWN proposed working group (Definitions for AI Workloads on the Network) is developing terminology and named entity definitions for AI workloads operating on network infrastructure.

Relationship: Composition at the registry layer. The SO Type Registry (draft-sato-soos-sov) is the operator-side implementation of the DAWN named entity concept: SO Types are versioned, signed, registry-governed definitions of the state machines that govern AI agent behavior. MAD extends the SO Type Registry to multi-agent workflows through SO-Type-Bound Creation Mandates (Section 3.3): an orchestrator with a creation mandate can instantiate cluster member SOs of that type, creating a distributed workflow topology derivable from the SO Type Registry definition alone.

Integration: DAWN named entity definitions SHOULD reference SO Type Registry entries as a concrete implementation. A SOOS SO Type Registry entry SHOULD be exportable as a DAWN-conforming named entity record.

15. Acknowledgements

The SO Instance Topology Types defined in Section 4 were developed through structured architectural review of the SOOS Cluster 1 Examination (Session 8). The SO Cluster coordination primitives in Section 5 were developed in dedicated OQ-S-28 resolution work (Session 10). The Narrowing Property builds on the delegation model work in draft-mw-spice-actor-chain and the AuthZEN 1.0 PEP/PDP separation model. The authors thank George Fletcher, Karl McGuinness, and the OAuth Working Group for Transaction Tokens and attenuating token work that informed the credential-layer analysis in Appendix B.

Author’s Address

Tom Sato
MyAuberge K.K.
Chino, Nagano, Japan
Email: tomsato@myauberge.jp
URI: https://activitytravel.pro/