

# **Authority Intermediate Representation (AIR) Specification Draft v0.1**

---

A Runtime Authority Language for Distributed Digital and AI Systems

Rob Caswell  
Authority Engineering / Authority Architecture V2  
March 2026

# Publication Record

---

## Authority Intermediate Representation (AIR): Specification Draft v0.1

**Author:** Rob Caswell

**Publisher:** Cyber Terrain LLC

**Published:** 2026-03-01

**Zenodo version:** v1.0

**DOI:** <https://doi.org/10.5281/zenodo.21742309>

**Website:** <https://cyberterrain.ai/papers/authority-intermediate-representation.pdf>

### Corpus Context

**Corpus:** Authority Engineering

**Document ID:** AE-IR-0001

**Object family:** AE-IR-0001

*Publisher-imprint edition prepared for versioned republication.*

*The prior Zenodo record remains the citable record until its new version is published.*

## Version History

| Version | Date       | Author      | Description                         |
|---------|------------|-------------|-------------------------------------|
| v0.1    | March 2026 | Rob Caswell | Initial working specification draft |

---

## Table of Contents

1. Document status
2. Abstract
3. Design goals
4. Conceptual model
5. Normative language
6. Canonical primitives
7. AIR architecture
8. AIR-S object model
9. AIR-S top-level schema
10. AIR-S generic node schema
11. AIR-S generic edge schema
12. AIR-S key invariants
13. AIR-R object model
14. AIR-R instruction classes
15. AIR-R universal requirements
16. Authority path model
17. Escalation chain model
18. AIR type system
19. Control safety ordering
20. Static admissibility laws
21. AIR compiler model

22. AIR decision engine
23. Runtime posture
24. Terminal model
25. Terminal lowering
26. Explanation model
27. Causal reason graph
28. Replay model
29. Diff model
30. Actual vs expected comparison
31. Canonical laws
32. Conformance
33. Minimal conformance profile
34. Open specification areas
35. Recommended next deliverables

## Appendices

- Appendix A — AIR Glossary
- Appendix B — AIR Opcode Reference
- Appendix C — AIR Decision Engine Pseudocode
- Appendix D — Authority Path Discovery Algorithm
- Appendix E — Authority Surface Telemetry Model
- Appendix F — Terminal Enforcement Contracts
- Appendix G — Authority Attacks and Failure Modes
- Appendix H — Example End-to-End Execution Trace
- Appendix I — AIR Formal Model
- Appendix J — AIR Reference Architecture

### 3. Motivation

Modern digital systems increasingly execute real-world consequences through automation, distributed services, and AI-driven decision processes. In these environments, authority is exercised not only through human commands but through signals transmitted between systems, workflows, and autonomous agents.

Traditional governance mechanisms such as identity access control, role-based permissions, and static policy enforcement are not sufficient for regulating these systems. Authority is now exercised dynamically through chains of automated actions that may cross multiple systems and consequence domains.

As a result, organizations face a new governance challenge:

How can governance intent be translated into executable constraints that safely regulate automated authority in distributed systems?

The Authority Intermediate Representation (AIR) addresses this challenge by introducing a machine-readable authority language that compiles governance and authority architecture into runtime semantics.

AIR enables systems to:

- represent authority relationships explicitly
- evaluate authority paths before consequences occur
- enforce governance boundaries near execution points
- maintain explainable and replayable authority decisions

By separating centralized authority semantics from distributed consequence enforcement, AIR allows governance intent to remain authoritative while still operating safely in highly distributed environments.

The AIR specification defines the structures, algorithms, and runtime instructions required to implement such a system.

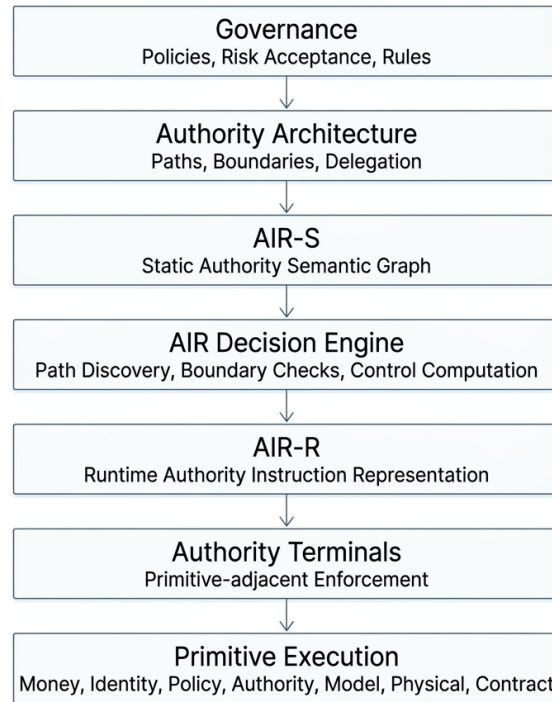


Figure 1 — AIR Authority Execution Stack

## AIR Specification Draft v0.1

### 1. Document status

Title: Authority Intermediate Representation (AIR) Specification Draft v0.1

Status: Working draft

Scope: Static authority semantics, runtime authority instructions, decision evaluation, distributed enforcement, explanation, and replay

Audience: Authority Architecture designers, control-plane engineers, terminal implementers, governance leads, runtime assurance teams

### 2. Abstract

AIR is a typed intermediate representation for authority systems in distributed digital and AI environments.

AIR compiles governance and authority architecture into machine-readable semantics that can be evaluated centrally and enforced near consequence execution points.

AIR is designed for systems where:

- actors emit signals that cause authorized systems to execute consequences
- attacks manipulate signals to trigger harmful actions
- automation and AI create synthetic insiders
- authority must be evaluated across consequence primitives:

- Money
- Identity
- Policy
- Authority
- Model
- Physical
- Contract

AIR consists of two linked representations:

- AIR-S: static authority semantic graph
- AIR-R: runtime authority instruction language

Together they enable:

- primitive truth binding
- authority path discovery
- escalation chain detection
- boundary evaluation
- admissible control selection
- central semantic control with distributed consequence enforcement
- replayable and explainable decision making

### 3. Design goals

AIR has eight core goals.

#### 3.1 Primitive truth preservation

AIR must classify the true consequence primitive, not trust app-layer labels.

#### 3.2 Authority path visibility

AIR must make authority paths explicit, including declared, derived, emergent, and illegitimate paths.

#### 3.3 Boundary-first enforcement

AIR must preserve mediation, review, threshold, safety, and crossing boundaries as first-class semantic objects.

#### 3.4 Distributed monotonic safety

Local terminals may tighten enforcement based on local truth, but may not broaden authority.

#### 3.5 Runtime evaluability

AIR must support bounded-latency decision making in live systems.

### 3.6 Replayability

Every material decision must be reproducible under the same semantic epoch and evidence set.

### 3.7 Explainability

Every material control action must be explainable as facts, inferences, admissibility, and selection.

### 3.8 Target agnosticism

AIR must lower into multiple terminal classes: API, workflow, identity, model, financial, and physical systems.

## 4. Conceptual model

# Authority Surface

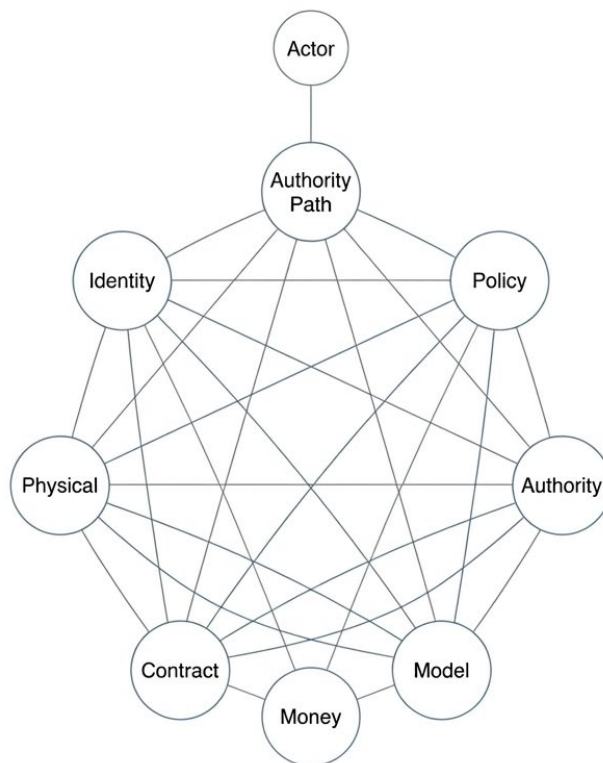


Figure 3 — Authority Surface

The authority surface represents the set of consequence primitives reachable from an actor through authority paths. Transitions between primitives define the possible consequence states that automated systems may induce.

AIR sits between governance and execution:

Governance

→ defines legitimacy, authority ownership, stop rights, mediation requirements

Authority Architecture

→ defines paths, zones, boundaries, terminals, delegation structure

AIR-S

→ compiles those semantics into a static graph

AIR decision engine

→ evaluates live requests against AIR-S and runtime state

AIR-R

→ emits runtime authority instructions and control decisions

Authority terminals

→ enforce at primitive-adjacent consequence points

## 5. Normative language

The keywords MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY are used in their ordinary specification sense.

## 6. Canonical primitives

AIR is built on primitive closure.

All consequence-bearing paths MUST terminate in one or more of these canonical primitives:

- MONEY
- IDENTITY
- POLICY
- AUTHORITY
- MODEL
- PHYSICAL
- CONTRACT

Subtypes MAY refine a primitive, but MUST NOT replace the primitive root for semantic evaluation.

## 7. AIR architecture

### AIR Distributed Enforcement Architecture

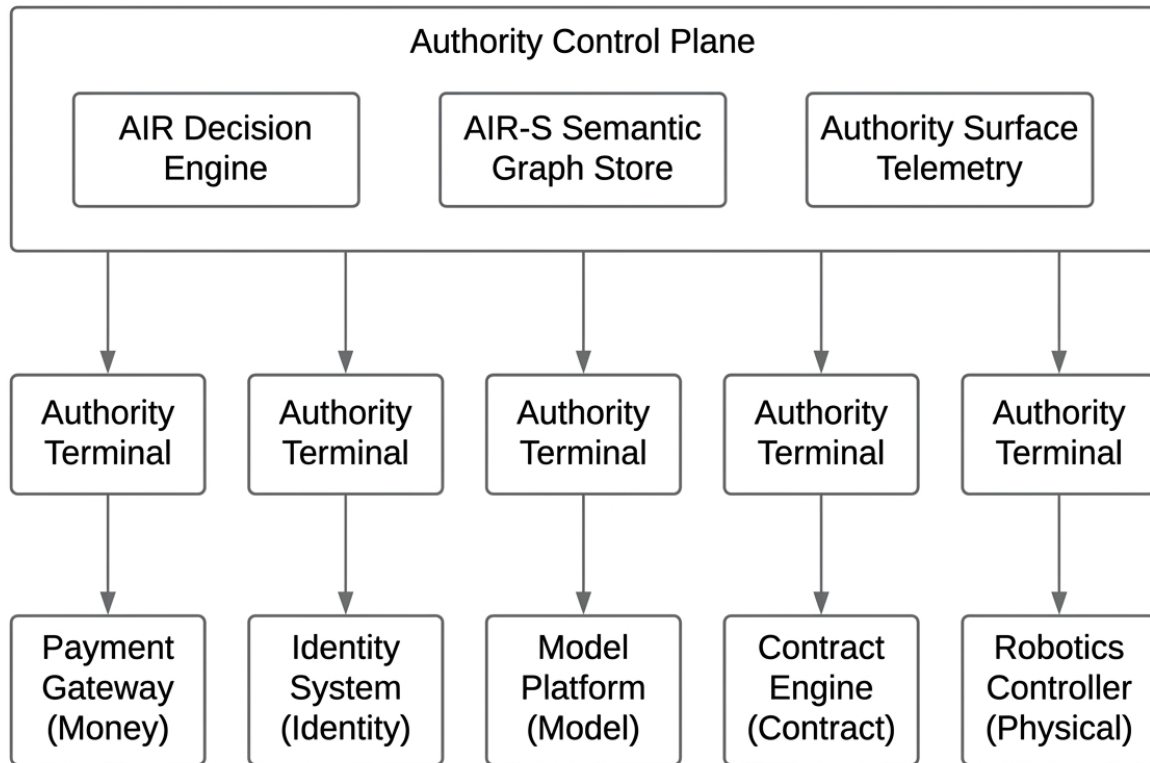


Figure 4 — AIR Distributed Enforcement Architecture

AIR centralizes authority semantics in the control plane while distributing enforcement to primitive-adjacent authority terminals. Terminals evaluate AIR-R instruction bundles and enforce controls near consequence execution systems.

AIR is a two-IR system.

#### 7.1 AIR-S

Static authority semantic graph.

Contains:

- primitive registry
- actor model
- zone model
- authority path graph
- boundary graph

- terminal model
- decision admissibility model

## 7.2 AIR-R

Runtime authority instruction language.

Contains:

- observation instructions
- transition instructions
- boundary instructions
- runtime-state instructions
- control instructions
- terminal results and telemetry

## 8. AIR-S object model

AIR-S is a typed labeled multigraph.

$G_{\{AIR-S\}} = (V, E, \tau_V, \tau_E, \lambda_V, \lambda_E)$

### 8.1 Node types

- Primitive
- Actor
- Zone
- Terminal
- Boundary
- Path
- DecisionPolicy

### 8.2 Edge types

- CAN\_TOUCH
- CAN\_TRANSITION
- DELEGATES\_TO
- REACHES
- CROSSES
- BOUNDED\_BY
- ENFORCED\_AT
- CONTAINED\_IN
- REQUIRES\_MEDIATION
- PROHIBITED

### 8.3 AIR-S modules

- primitive\_registry
- actor\_model
- zone\_model
- path\_graph
- boundary\_graph
- terminal\_model
- decision\_model

### 9. AIR-S top-level schema

```
{
  "airs_version": "0.1.0",
  "semantic_epoch": "2026.03.07.1",
  "environment_id": "prod-us-east",
  "compiled_from": {
    "governance_version": "gov-12",
    "architecture_version": "arch-7"
  },
  "metadata": {},
  "modules": {
    "primitive_registry": {},
    "actor_model": {},
    "zone_model": {},
    "path_graph": {},
    "boundary_graph": {},
    "terminal_model": {},
    "decision_model": {}
  },
  "integrity": {}
}
```

### 10. AIR-S generic node schema

```
{
  "node_id": "string",
  "node_type": "Primitive|Actor|Zone|Terminal|Boundary|Path|DecisionPolicy",
  "label": "string",
  "type_ref": "string",
  "attributes": {},
  "tags": ["string"],
  "status": "ACTIVE|DEPRECATED|DISABLED"
}
```

### 11. AIR-S generic edge schema

```
{
  "edge_id": "string",
  "edge_type": "string",
  "from_ref": {
    "ref_type": "NODE|EDGE",

```

```

    "ref_id": "string"
  },
  "to_ref": {
    "ref_type": "NODE|EDGE",
    "ref_id": "string"
  },
  "attributes": {},
  "conditions": {},
  "tags": ["string"],
  "status": "ACTIVE|DEPRECATED|DISABLED"
}

```

## 12. AIR-S key invariants

### 12.1 Primitive closure

All executable consequence paths **MUST** resolve to canonical primitives.

### 12.2 Boundary preservation

All source-declared mandatory boundaries **MUST** remain represented in AIR-S.

### 12.3 Terminal coverage

Every executable high-consequence primitive path **MUST** map to at least one enforcing terminal.

### 12.4 Projection safety

Any AIR-S projection **MAY** reduce scope, but **MUST NOT** omit semantically required boundaries for the projected artifact.

### 12.5 Monotonic terminal constraints

Terminal capability definitions **MUST NOT** authorize local semantic broadening.

## 13. AIR-R object model

AIR-R is the runtime instruction language.

Each instruction has this top-level form:

```

{
  "air_version": "0.1",
  "instruction_id": "air-uuid",
  "bundle_id": "bundle-uuid",
  "timestamp": "RFC3339 timestamp",
  "instruction_class":
  "Observation|Transition|Boundary|RuntimeState|Control",

```

```

"opcode": "string",
"issuer": "string",
"target_terminal": "string|null",
"actor": {},
"action": {},
"primitive": {},
"authority_path": {},
"boundary_context": {},
"runtime_state": {},
"control": {},
"telemetry": {},
"integrity": {}
}

```

## 14. AIR-R instruction classes

### 14.1 Observation

Declares runtime facts without authorizing consequence execution.

Examples:

- ACTOR\_DECLARE
- ACTION\_REQUEST
- PRIMITIVE\_TOUCH
- ZONE\_MEMBERSHIP\_ASSERT
- REACHABILITY\_EXPAND
- PRESSURE\_SPIKE
- SYNC\_STALE

### 14.2 Transition

Represents attempted consequence-bearing changes.

Examples:

- PRIMITIVE\_MUTATE
- FUNDS\_TRANSFER
- IDENTITY\_BIND
- POLICY\_CHANGE
- AUTHORITY\_ASSIGN
- MODEL\_DEPLOY
- PHYSICAL\_ACTUATE
- CONTRACT\_BIND

### 14.3 Boundary

Represents crossings, checks, and violations.

Examples:

- BOUNDARY\_CHECK
- BOUNDARY\_VIOLATION
- ESCALATION\_CROSS
- ZONE\_EXIT\_ATTEMPT
- THRESHOLD\_EXCEEDED
- DELEGATION\_DEPTH\_EXCEEDED

#### 14.4 RuntimeState

Represents live authority posture changes.

Examples:

- POLICY\_VERSION\_MISMATCH
- TERMINAL\_ISOLATED
- MEDIATION\_PENDING
- ANOMALY\_ELEVATED
- LOCAL\_SAFETY\_ELEVATION
- RISK\_ACCEPTED\_CONTEXT
- STATE\_CONVERGENCE\_DETECTED

#### 14.5 Control

Represents executable enforcement directives.

Examples:

- ALLOW
- DENY
- THROTTLE
- BUFFER
- VERIFY
- PARTITION
- MEDIATION\_REQUIRED
- TERMINATE
- ACCEPT\_RISK

### 15. AIR-R universal requirements

Every AIR-R instruction MUST contain:

- air\_version
- instruction\_id
- bundle\_id
- timestamp
- instruction\_class

- opcode
- issuer
- integrity

All consequence-bearing Transition instructions **MUST** also contain:

- actor
- action
- primitive
- authority\_path.path\_id
- authority\_path.path\_completeness

All Control instructions **MUST** contain:

- control.decision
- control.decision\_scope
- control.issued\_by
- control.issued\_at
- valid integrity protection

## 16. Authority path model

An authority path is the causally ordered semantic route by which an actor can induce a consequence at a primitive node.

AIR path semantics **MUST** track:

- path\_kind
- path\_completeness
- path\_validity
- primitive\_chain
- delegation\_chain
- reachability\_basis
- escalation\_depth

### 16.1 Path kinds

- DECLARED
- DERIVED
- EMERGENT
- ILLEGITIMATE

### 16.2 Path completeness

- COMPLETE
- PARTIAL
- UNKNOWN
- INVALID

### 16.3 Path validity

- VALID
- DEGRADED
- STALE
- INVALID
- UNDER\_MEDIATION

### 17. Escalation chain model

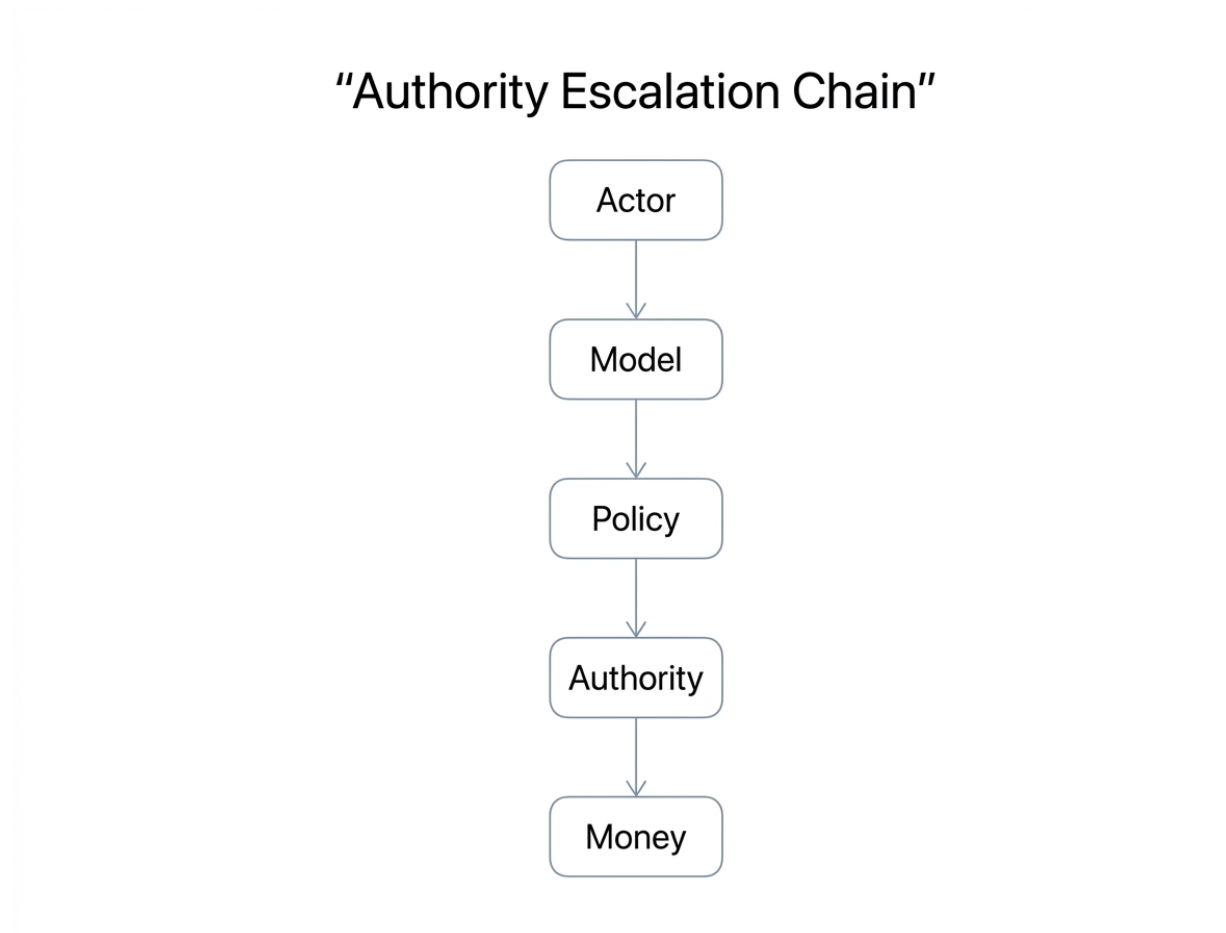


Figure 5 — Authority Escalation Chain

An actor's request may propagate through multiple authority primitives before reaching a high-impact consequence. Automated systems frequently exhibit escalation paths where model outputs influence policy evaluation, policy decisions trigger authorization, and authorization ultimately executes financial or physical consequences.

An escalation chain is an ordered transition sequence in which one primitive action creates, unlocks, or expands reachable consequence against another primitive or authority scope.

AIR MUST detect escalation-significant conditions including:

- first-time primitive crossing
- delegation depth increase with new consequence scope
- tool-driven reachability expansion
- model-to-policy or policy-to-authority transition
- zone exit with increased exposure
- path compression that reduces oversight distance

## 18. AIR type system

AIR uses typed domains for static and runtime correctness.

### 18.1 Primitive types

- MONEY
- IDENTITY
- POLICY
- AUTHORITY
- MODEL
- PHYSICAL
- CONTRACT

### 18.2 Consequence magnitude

- LOW
- MODERATE
- HIGH
- CRITICAL

### 18.3 Irreversibility

- REVERSIBLE
- PARTIAL
- EFFECTIVELY\_IRREVERSIBLE
- IRREVERSIBLE

### 18.4 Actor types

- HUMAN
- SERVICE
- WORKFLOW
- AI\_AGENT
- MULTI\_AGENT\_COORDINATOR
- PROXY
- SYSTEM

### 18.5 Actor subtypes

- STANDARD

- SYNTHETIC\_INSIDER
- DELEGATED\_PROXY
- EXECUTION\_AGENT
- AUDIT\_AGENT
- ARBITER\_AGENT

## 18.6 Zone modes

- CONTAINED
- SUPERVISED
- OPEN
- RESTRICTED
- ISOLATED

## 18.7 Sync states

- FRESH
- DEGRADED
- STALE
- ISOLATED

## 18.8 Control types

- ALLOW
- DENY
- THROTTLE
- BUFFER
- VERIFY
- PARTITION
- MEDIATION\_REQUIRED
- TERMINATE
- ACCEPT\_RISK

## 19. Control safety ordering

AIR defines a monotonic safety ordering:

ALLOW < THROTTLE < BUFFER < VERIFY < MEDIATION\_REQUIRED < DENY < TERMINATE

ACCEPT\_RISK is an exception wrapper, not a weaker ALLOW.

This ordering governs local tightening rules.

## 20. Static admissibility laws

### 20.1 No illegitimate allow

ILLEGITIMATE paths MUST NOT produce ALLOW.

## 20.2 No incomplete high-consequence allow

Incomplete high-consequence consequence paths MUST NOT auto-release.

## 20.3 No emergent high-consequence allow

If path\_kind = EMERGENT and magnitude is HIGH or CRITICAL, ALLOW MUST be inadmissible.

## 20.4 No stale irreversible optimistic allow

For irreversible or effectively irreversible consequences under insufficient runtime confidence, permissive decisions MUST tighten.

## 20.5 No unsigned material control

Material control decisions MUST be integrity-protected.

## 21. AIR compiler model

AIR compilation proceeds through these layers:

Governance intent

- Authority Architecture
- AIR-S
- AIR decision engine
- AIR-R
- terminal lowering

### 21.1 Compiler preservation goals

Compilation MUST preserve:

- primitive truth
- boundary meaning
- path legitimacy
- mediation requirements
- stop rights
- monotonic terminal limits

### 21.2 Forbidden compiler degradations

Compilation MUST NOT:

- erase true primitive identity
- collapse required mediation into silent allow
- remove mandatory boundaries
- weaken stop authority

- broaden terminal authority

## 22. AIR decision engine

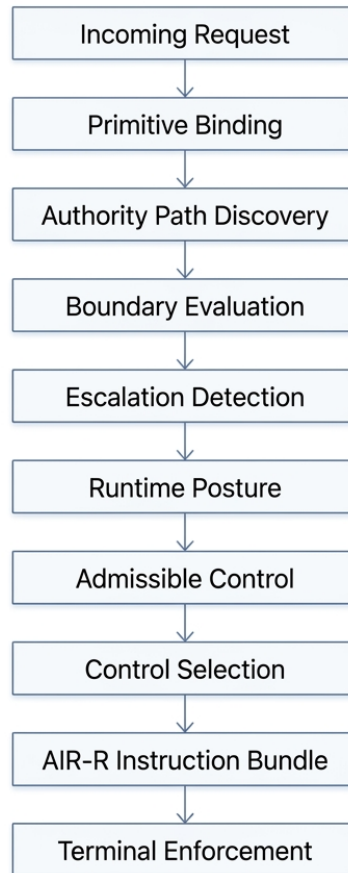


Figure 2 — AIR Runtime Decision Pipeline

The decision engine evaluates live requests against AIR-S and runtime state.

Decision function:

$\text{Decide}(\Omega) \rightarrow \langle A, d, B, X \rangle$

Where:

- = admissible control set
- = selected control
- = AIR-R bundle
- = explanation object

## 22.1 Decision phases

1. ingestion
2. normalization
3. primitive binding
4. actor and zone resolution
5. path discovery
6. boundary attachment
7. runtime posture synthesis
8. admissibility computation
9. control selection
10. AIR-R emission
11. terminal lowering
12. explanation and replay capture

## 22.2 Decision law

The engine **MUST** choose the least-dangerous admissible control that preserves authority semantics and runtime safety.

## 23. Runtime posture

The decision engine **MUST** synthesize runtime posture from:

- sync state
- terminal health
- anomaly
- pressure
- convergence
- isolation
- risk acceptance scope

Derived runtime confidence:

- SUFFICIENT
- DEGRADED
- INSUFFICIENT

Runtime posture **MUST** constrain admissibility.

## 24. Terminal model

Authority terminals are primitive-adjacent enforcement points.

### 24.1 Terminal authority classes

- ENFORCE\_ONLY
- ENFORCE\_AND\_VERIFY

- ENFORCE\_AND\_TIGHTEN

## 24.2 Terminal rule

No AIR terminal MAY broaden central semantics.

## 24.3 Local tightening

A terminal MAY tighten only if:

- terminal class allows it
- local runtime truth materially changed
- resulting decision moves rightward in safety ordering
- resulting scope is narrowed or equal
- event is logged as LOCAL\_SAFETY\_ELEVATION

## 25. Terminal lowering

AIR controls MUST lower into terminal-specific enforcement actions.

Examples:

- ALLOW → execute
- DENY → block
- THROTTLE → rate-limit
- BUFFER → queue/delay
- VERIFY → proof/attestation gate
- PARTITION → isolate scope
- MEDIATION\_REQUIRED → hard block plus adjudication route
- TERMINATE → immediate stop
- ACCEPT\_RISK → bounded exception release

A control that cannot be faithfully enforced at the relevant terminal MUST be treated as invalid and re-routed, tightened, or denied.

## 26. Explanation model

Every material control decision MUST produce an explanation object.

Explanation MUST include:

- semantic epoch
- primitive truth
- actor state
- zone state
- path state
- boundary outcomes
- runtime posture
- admissible controls

- selected control
- reason graph
- terminal outcome

## 26.1 Explanation layers

- facts
- inferences
- admissibility effects
- final selection
- terminal outcome

## 27. Causal reason graph

Explanations SHOULD represent causality as a reason graph.

Reason node types:

- FACT
- BOUNDARY
- INFERENCE
- ADMISSIBILITY
- SELECTION

Reason relations:

- SUPPORTS
- CONSTRAINS
- INVALIDATES
- TRIGGERS
- DERIVES
- LEADS\_TO
- OVERRIDES

## 28. Replay model

AIR replay reconstructs a prior decision using:

- original AIR-S semantic epoch
- original event/request
- original runtime evidence
- original local context
- original integrity state

Replay modes:

- EXACT
- COUNTERFACTUAL
- EPOCH

Replay MUST preserve semantic epoch unless explicitly operating in epoch-diff mode.

## 29. Diff model

AIR diff compares:

- AIR-S epoch to AIR-S epoch
- decision to decision
- expected terminal outcome to actual terminal outcome

Diff MUST identify impact on:

- admissible control space
- path classification
- boundary application
- terminal enforcement behavior

## 30. Actual vs expected comparison

AIR MUST support comparison between:

- expected central decision
- actual terminal behavior

Verdicts:

- MATCH
- MONOTONIC\_TIGHTENING
- SEMANTIC\_DEVIATION
- ENFORCEMENT\_FAILURE
- INSUFFICIENT\_EVIDENCE

## 31. Canonical laws

These are the core AIR laws.

### Law 1: Primitive closure

All consequence paths terminate in canonical primitives.

### Law 2: Primitive truth precedence

True primitive type outranks app-layer naming.

### Law 3: Path-before-permission

Authority path legitimacy must be evaluated before consequence release.

#### Law 4: Boundary-first authority control

Mandatory boundaries are first-class and non-elidable.

#### Law 5: No emergent high-consequence auto-release

Emergent high-consequence paths cannot silently execute.

#### Law 6: Mediation preservation

Source-declared mediation cannot be compiled away.

#### Law 7: No unsigned material control

Material consequence controls require integrity protection.

#### Law 8: Monotonic distributed safety

Local terminals may tighten, never broaden.

#### Law 9: Compression significance

Reduced oversight distance is semantically meaningful even when primitive reach is unchanged.

#### Law 10: Replayability

Every material decision must be reproducible or explainably non-reproducible under preserved evidence.

### 32. Conformance

#### 32.1 AIR-S conformance

An AIR-S implementation conforms if it:

- implements the AIR-S graph structure
- preserves required node and edge semantics
- validates required invariants
- supports safe projection
- provides integrity and version metadata

#### 32.2 AIR-R conformance

An AIR-R implementation conforms if it:

- emits well-formed typed instructions

- preserves required opcode semantics
- enforces required field and integrity rules
- respects mandatory emission relationships

### 32.3 Decision engine conformance

A decision engine conforms if it:

- binds primitive truth
- evaluates path legitimacy
- attaches relevant boundaries
- computes admissible controls before selection
- preserves monotonic terminal constraints
- produces replayable explanation artifacts

### 32.4 Terminal conformance

A terminal conforms if it:

- enforces only supported controls
- verifies integrity where required
- never broadens authority
- logs local tightening
- emits terminal outcomes for audit and replay

## 33. Minimal conformance profile

A minimum deployable AIR profile MAY implement:

### AIR-S

- primitive registry
- path graph
- boundary graph
- terminal model
- decision model

### AIR-R

- ACTOR\_DECLARE
- ACTION\_REQUEST
- PRIMITIVE\_TOUCH
- ESCALATION\_CROSS
- BOUNDARY\_CHECK
- SYNC\_STALE
- ALLOW
- DENY
- MEDIATION\_REQUIRED

## Decision engine

- primitive binding
- path discovery
- boundary evaluation
- admissibility computation
- control selection
- explanation output

That is sufficient to prove the architecture.

## 34. Open specification areas

The following areas remain intentionally open in v0.1:

- formal JSON Schema publication for AIR-R and AIR-S
- full opcode-by-opcode machine-checkable contracts
- canonical query language for AIR-S traversal
- policy-pack packaging format for terminals
- formal proof obligations for compiler preservation theorems
- benchmark suite for replay and semantic diffing

## 35. Recommended next deliverables

The strongest next deliverables are:

1. AIR glossary and terminology appendix
2. AIR-S JSON Schema draft
3. AIR-R JSON Schema draft
4. Opcode reference appendix
5. Decision engine pseudocode appendix
6. Worked conformance traces appendix
7. Threat and failure modes appendix

The next best move is the appendix set, starting with the glossary and opcode reference.

## Appendix A — AIR Glossary

### Authority System

A distributed system in which actors emit signals that cause authorized mechanisms to execute real-world or digital consequences.

### Authority Surface

The total set of consequence-changing state transitions reachable within a system. AIR evaluates requests against this surface to determine whether transitions are legitimate.

### Authority Path

The ordered causal chain by which an actor signal can produce a primitive consequence.

### Escalation Chain

A sequence of transitions where one primitive unlocks or expands reachability to another primitive or authority scope.

### Primitive

The irreducible consequence class affected by an action. AIR defines seven canonical primitives: MONEY, IDENTITY, POLICY, AUTHORITY, MODEL, PHYSICAL, CONTRACT.

### Primitive Closure

The rule that every consequence-bearing action must terminate in one or more canonical primitives.

### Synthetic Insider

An automated actor (AI agent, workflow, service) operating with legitimate authority inside the authority surface.

### Authority Boundary

A structural constraint that must be crossed or satisfied before authority can progress along a path.

### Mediation Boundary

A boundary requiring human or adjudicated approval before execution continues.

### Containment Zone

A bounded region of the authority surface in which automated actors may operate without escalation to higher-consequence primitives.

### Authority Terminal

A distributed enforcement point located near consequence execution.

### Authority Control Plane

The central authority engine responsible for evaluating AIR instructions and issuing enforcement decisions.

## AIR-S

The static semantic authority graph compiled from governance and architecture definitions.

## AIR-R

The runtime instruction representation used to describe live authority events and enforcement decisions.

## AIR Instruction

A structured runtime message representing observation, transition, boundary, runtime state, or control semantics.

## Opcode

The specific operation represented by an AIR instruction.

## Admissible Control Set

The set of enforcement actions permitted after evaluating authority paths, boundaries, and runtime state.

## Runtime Posture

The synthesized evaluation of system health, synchronization state, anomaly levels, and terminal conditions used during decision making.

## Semantic Epoch

The versioned snapshot of AIR-S semantics used for runtime decision evaluation and replay.

## Monotonic Safety

The rule that distributed enforcement points may tighten authority but never broaden it relative to central semantics.

## Control Instruction

An AIR instruction directing enforcement behavior at a terminal.

## Reason Graph

A structured explanation artifact capturing facts, inferences, admissibility constraints, and final control decisions.

## Replay

The deterministic reconstruction of a prior decision using the same semantic epoch and evidence set.

## Terminal Tightening

A local enforcement decision that moves rightward in the AIR safety ordering due to local truth.

## Appendix B — AIR Opcode Reference

This appendix defines the canonical opcode set for AIR-R instructions. Opcodes are grouped by instruction class and define the semantic meaning of runtime authority events and control actions.

### B.1 Observation Opcodes

Observation instructions declare facts about runtime authority context. They do not authorize consequences.

#### ACTOR\_DECLARE

Declares the identity and type of an actor participating in the authority system.

Required fields:

- `actor.actor_id`
- `actor.actor_type`
- `actor.actor_subtype`

#### ACTION\_REQUEST

Represents a request by an actor to perform an action that may affect a primitive.

Required fields:

- `actor`
- `action`

#### PRIMITIVE\_TOUCH

Indicates that an action intends to interact with a primitive consequence class.

Required fields:

- `primitive.primitive_type`
- `action.action_type`

## ZONE\_MEMBERSHIP\_ASSERT

Asserts that an actor or workflow is operating inside a specific containment zone.

Required fields:

- `actor.actor_id`
- `zone.zone_id`

## REACHABILITY\_EXPAND

Signals that new reachable authority paths have been discovered, typically due to new tool access, integration, or delegation.

Required fields:

- `actor.actor_id`
- `authority_path.reachability_basis`

## PRESSURE\_SPIKE

Signals rapid growth in authority flow or execution demand toward a primitive.

Required fields:

- `primitive.primitive_type`
- `runtime_state.pressure_level`

## SYNC\_STALE

Indicates that the local terminal or control plane is operating with degraded semantic synchronization.

Required fields:

- `runtime_state.sync_state`

---

## B.2 Transition Opcodes

Transition instructions represent attempted consequence-bearing changes.

## PRIMITIVE\_MUTATE

Generic primitive state change.

## FUNDS\_TRANSFER

Represents transfer of value within the MONEY primitive.

Required fields:

- primitive: MONEY
- action.amount
- action.destination

## IDENTITY\_BIND

Associates credentials, roles, or identities.

Required fields:

- primitive: IDENTITY

## POLICY\_CHANGE

Represents modification of rules governing system behavior.

Required fields:

- primitive: POLICY

## AUTHORITY\_ASSIGN

Represents delegation or reassignment of authority rights.

Required fields:

- primitive: AUTHORITY

## MODEL\_DEPLOY

Represents deployment or modification of a machine-learning or AI model.

Required fields:

- primitive: MODEL

## PHYSICAL\_ACTUATE

Represents a command that causes physical-world consequences.

Required fields:

- primitive: PHYSICAL

## CONTRACT\_BIND

Represents the creation or modification of legally binding obligations.

Required fields:

- primitive: CONTRACT
- 

## B.3 Boundary Opcodes

Boundary instructions represent constraint evaluation.

### BOUNDARY\_CHECK

Evaluates whether a boundary condition is satisfied.

Required fields:

- boundary\_context.boundary\_id

### BOUNDARY\_VIOLATION

Signals that a mandatory boundary has been violated.

Required fields:

- boundary\_context.boundary\_id

### ESCALATION\_CROSS

Indicates that an action crosses an escalation boundary.

Required fields:

- authority\_path.escalation\_depth

### ZONE\_EXIT\_ATTEMPT

Signals attempted execution outside a containment zone.

Required fields:

- zone.zone\_id

### THRESHOLD\_EXCEEDED

Signals crossing of predefined risk or magnitude thresholds.

Required fields:

- `runtime_state.threshold_id`

### DELEGATION\_DEPTH\_EXCEEDED

Signals delegation chains exceeding allowed authority depth.

Required fields:

- `authority_path.delegation_chain`

---

## B.4 Runtime State Opcodes

Runtime state instructions represent posture changes affecting decision logic.

### POLICY\_VERSION\_MISMATCH

The terminal or actor is operating against a policy version inconsistent with the control plane.

### TERMINAL\_ISOLATED

Terminal has lost control-plane connectivity.

### MEDIATION\_PENDING

A mediation process is underway.

### ANOMALY\_ELEVATED

An anomaly score or risk signal has exceeded acceptable levels.

### LOCAL\_SAFETY\_ELEVATION

Terminal has locally tightened enforcement due to runtime conditions.

### RISK\_ACCEPTED\_CONTEXT

An explicit risk acceptance scope is currently active.

## STATE\_CONVERGENCE\_DETECTED

Multiple authority paths are converging toward a single primitive.

---

### B.5 Control Opcodes

Control instructions direct enforcement actions.

#### ALLOW

Permits the requested action to execute.

#### DENY

Blocks the requested action.

#### THROTTLE

Limits execution rate.

#### BUFFER

Queues execution until conditions improve.

#### VERIFY

Requires attestation, verification, or proof.

#### PARTITION

Restricts execution scope or isolates the request.

#### MEDIATION\_REQUIRED

Routes the action to adjudication or human review.

#### TERMINATE

Immediately halts the execution path.

#### ACCEPT\_RISK

Explicitly allows execution despite elevated risk conditions under a defined scope.

---

## B.6 Opcode stability

AIR opcode identifiers are stable semantic identifiers and **MUST NOT** change meaning across semantic epochs. New opcodes **MAY** be added but existing ones **MUST** preserve backward semantic compatibility.

## Appendix C — AIR Decision Engine Pseudocode

This appendix defines a reference decision algorithm for evaluating runtime authority events using AIR-S semantics and producing AIR-R control instructions.

The algorithm is descriptive rather than prescriptive. Implementations **MAY** optimize internally but **MUST** preserve semantic equivalence with the defined phases.

### C.1 Decision Inputs

The decision engine operates over the tuple:

$\Omega = (\text{event}, \text{AIR-S}, \text{runtime\_context})$

Where:

- event = incoming AIR-R observation or transition request
- AIR-S = compiled semantic authority graph
- runtime\_context = live system posture including sync state, anomaly signals, and terminal health

### C.2 High-Level Algorithm

function Decide(event, AIR-S, runtime\_context):

```

    normalized_event = normalize(event)

    primitive = bind_primitive(normalized_event)

    actor_state = resolve_actor(normalized_event.actor, AIR-S)

    zone_state = resolve_zone(actor_state, AIR-S)

    candidate_paths = discover_paths(actor_state, primitive, AIR-S)

    path_state = evaluate_paths(candidate_paths)

    boundary_results = evaluate_boundaries(path_state, AIR-S)

    escalation_state = detect_escalation(path_state)
```

```
runtime_posture = synthesize_runtime_posture(runtime_context)

admissible_controls = compute_admissible_controls(
    primitive,
    path_state,
    boundary_results,
    escalation_state,
    runtime_posture
)

decision = select_control(admissible_controls)

bundle = emit_air_bundle(event, decision, path_state)

explanation = generate_reason_graph(
    event,
    primitive,
    path_state,
    boundary_results,
    runtime_posture,
    admissible_controls,
    decision
)

return (decision, bundle, explanation)
```

---

### C.3 Primitive Binding

Primitive binding determines the true consequence class of an action.

```
function bind_primitive(event):

    declared = event.primitive

    inferred = infer_primitive_from_action(event.action)

    if inferred != null:
        return inferred

    return declared
```

Primitive truth MUST override application labels when evidence indicates a different primitive.

---

## C.4 Authority Path Discovery

Authority path discovery determines how an actor could legitimately produce the requested primitive transition.

```
function discover_paths(actor, primitive, AIR-S):  
    paths = []  
    for edge in AIR-S.path_graph:  
        if edge.from == actor and edge.reaches_primitive == primitive:  
            paths.append(edge)  
    return paths
```

Paths are classified as:

- DECLARED
- DERIVED
- EMERGENT
- ILLEGITIMATE

---

## C.5 Boundary Evaluation

Boundaries are evaluated against the discovered authority paths.

```
function evaluate_boundaries(paths, AIR-S):  
    results = []  
    for path in paths:  
        for boundary in path.boundaries:  
            result = check_boundary(boundary)  
            results.append(result)  
    return results
```

Boundary checks MAY produce:

- satisfied
- pending mediation
- violated

## C.6 Escalation Detection

Escalation occurs when transitions expand reachable authority or cross primitive domains.

```
function detect_escalation(path_state):  
    if path_state.crosses_primitive:  
        return ESCALATION_CROSS  
  
    if path_state.delegation_depth_increase:  
        return ESCALATION_CROSS  
  
    if path_state.zone_exit:  
        return ESCALATION_CROSS  
  
    return NONE
```

Escalation detection influences admissible control computation.

---

## C.7 Runtime Posture Synthesis

Runtime posture aggregates environmental signals affecting authority safety.

```
function synthesize_runtime_posture(runtime_context):  
    posture = {}  
  
    posture.sync_state = runtime_context.sync_state  
    posture.anomaly = runtime_context.anomaly  
    posture.pressure = runtime_context.pressure  
    posture.terminal_health = runtime_context.terminal_health  
  
    if posture.sync_state == STALE:  
        posture.confidence = INSUFFICIENT  
  
    else if posture.anomaly > threshold:  
        posture.confidence = DEGRADED  
  
    else:  
        posture.confidence = SUFFICIENT  
  
    return posture
```

---

## C.8 Admissible Control Computation

The admissible control set is computed using path legitimacy, boundaries, primitive severity, and runtime posture.

```
function compute_admissible_controls(  
    primitive,  
    path_state,  
    boundary_results,  
    escalation_state,  
    runtime_posture  
)  
:  
  
    controls = {}  
  
    if path_state.kind == ILLEGITIMATE:  
        controls = {DENY, TERMINATE}  
  
    else if boundary_results.contains_violation:  
        controls = {DENY, MEDIATION_REQUIRED}  
  
    else if runtime_posture.confidence == INSUFFICIENT:  
        controls = {BUFFER, VERIFY, DENY}  
  
    else:  
        controls = {ALLOW, THROTTLE, VERIFY}  
  
    return controls
```

---

## C.9 Control Selection

The decision engine selects the least-dangerous admissible control according to the AIR safety ordering.

```
function select_control(admissible_controls):  
  
    ordered = sort_by_safety_order(admissible_controls)  
  
    return ordered[0]
```

Safety ordering is defined in Section 19.

---

## C.10 AIR Bundle Emission

After selecting a decision, the engine emits a bundle of AIR instructions representing the evaluation trace.

```
function emit_air_bundle(event, decision, path_state):  
    bundle = []  
    bundle.append(event)  
    bundle.append({opcode: PRIMITIVE_TOUCH})  
    bundle.append({opcode: BOUNDARY_CHECK})  
    bundle.append({opcode: decision})  
    return bundle
```

---

### C.11 Explanation Generation

The explanation system constructs a reason graph capturing the causal chain behind the decision.

```
function generate_reason_graph(...):  
    graph = ReasonGraph()  
    graph.add_fact("primitive", primitive)  
    graph.add_fact("path_kind", path_state.kind)  
    graph.add_fact("runtime_confidence", runtime_posture.confidence)  
    graph.add_decision(decision)  
    return graph
```

The explanation artifact **MUST** be preserved for replay and audit.

---

### C.12 Determinism Requirement

For a fixed semantic epoch and evidence set, the decision engine **MUST** produce the same decision outcome.

If determinism cannot be preserved, the explanation **MUST** identify the nondeterministic inputs.

## Appendix D — Authority Path Discovery Algorithm

This appendix defines the reference algorithm for discovering authority paths between actors and consequence primitives within the AIR-S graph.

Authority path discovery determines how an actor signal could legitimately or illegitimately cause a primitive transition.

The algorithm operates on the AIR-S semantic graph and produces candidate authority paths used by the decision engine.

---

### D.1 Path Discovery Inputs

Path discovery operates over the tuple:

$\Psi = (\text{actor}, \text{primitive}, \text{AIR-S})$

Where:

- actor = requesting actor node
- primitive = target consequence primitive
- AIR-S = authority semantic graph

---

### D.2 Path Discovery Goals

The algorithm MUST identify:

- all reachable primitive paths
- delegation chains
- primitive transitions
- boundary crossings
- escalation events

The algorithm MUST also classify each discovered path as:

- DECLARED
- DERIVED
- EMERGENT
- ILLEGITIMATE

---

### D.3 Graph Traversal Model

AIR path discovery uses constrained graph traversal over AIR-S.

Traversal follows edges:

- CAN\_TOUCH
- CAN\_TRANSITION

- DELEGATES\_TO
- REACHES

Traversal MUST respect boundary constraints encoded in the boundary graph.

Traversal MUST NOT cross PROHIBITED edges.

---

#### D.4 Reference Path Discovery Algorithm

```
function discover_paths(actor, primitive, AIR-S):  
    paths = []  
    frontier = [{node: actor, path: []}]  
    visited = set()  
    while frontier not empty:  
        current = frontier.pop()  
        node = current.node  
        path = current.path  
        if node == primitive:  
            paths.append(path)  
            continue  
        if node in visited:  
            continue  
        visited.add(node)  
        for edge in outgoing_edges(node):  
            if edge.type == PROHIBITED:  
                continue  
            next_node = edge.target  
            new_path = path + [edge]  
            frontier.push({node: next_node, path: new_path})  
    return classify_paths(paths)
```

---

## D.5 Path Classification

Discovered paths are classified according to their relationship to declared architecture.

### DECLARED

A path explicitly defined in AIR-S via path\_graph declarations.

### DERIVED

A path not explicitly declared but logically implied by delegation or transition edges.

### EMERGENT

A path created by runtime integrations, tool access, or cross-system connectivity not originally intended in architecture design.

### ILLEGITIMATE

A path that violates architectural constraints or crosses prohibited edges.

---

## D.6 Escalation Detection During Discovery

During traversal, the algorithm MUST detect escalation events including:

- primitive domain crossings
- new delegation authority
- containment zone exit
- path compression

If escalation is detected, the path MUST be flagged with escalation metadata.

---

## D.7 Delegation Chain Tracking

Authority delegation chains are captured as ordered actor transitions.

$A \rightarrow B \rightarrow C \rightarrow \text{primitive}$

Delegation metadata MUST include:

- delegation depth
- delegating actor
- delegated authority scope

Delegation depth MAY be constrained by policy.

---

## D.8 Boundary Attachment

Boundaries encountered along a path MUST be attached to the path representation.

Example path representation:

```
Actor
  → CAN_TOUCH
Service
  → DELEGATES_TO
Workflow
  → BOUNDARY
Human Approval
  → CAN_TRANSITION
MONEY
```

Boundaries are evaluated later by the decision engine.

---

## D.9 Path Compression Detection

Path compression occurs when automation reduces the number of mediation or oversight steps between actor and primitive.

Compression detection rule:

```
if new_path_length < historical_path_length:
    mark_compression(path)
```

Compressed paths MUST trigger elevated scrutiny.

---

## D.10 Authority Surface Expansion Detection

Authority surface expansion occurs when new primitives become reachable from an actor.

Detection rule:

```
if reachable_primitives_new > reachable_primitives_previous:
    emit REACHABILITY_EXPAND
```

Expansion signals are emitted as AIR observation instructions.

## D.11 Path Output Structure

Each discovered path is represented as:

```
{
  "path_id": "string",
  "path_kind": "DECLARED|DERIVED|EMERGENT|ILLEGITIMATE",
  "primitive_chain": ["IDENTITY", "AUTHORITY", "MONEY"],
  "delegation_chain": ["actorA", "serviceB", "workflowC"],
  "boundaries": ["boundary1", "boundary2"],
  "escalation_depth": 1,
  "compressed": false
}
```

---

## D.12 Completeness Requirement

The discovery algorithm **MUST** enumerate all reachable primitive paths unless bounded by configured search limits.

If limits truncate discovery, the path state **MUST** be marked PARTIAL.

## Appendix E — Authority Surface Telemetry Model

This appendix defines the telemetry layer used by AIR to observe and measure the dynamic behavior of authority systems. Telemetry captures runtime signals describing how authority flows across primitives, paths, and terminals.

Authority Surface telemetry enables detection of:

- authority pressure
- convergence events
- reachability expansion
- escalation chains
- containment zone violations

Telemetry is emitted as AIR observation instructions and aggregated by the authority control plane.

---

## E.1 Telemetry Goals

The telemetry subsystem **MUST** support the following capabilities:

- continuous observation of authority flow
- detection of structural authority changes

- identification of escalation chains
- detection of abnormal authority pressure
- measurement of authority surface expansion

Telemetry **MUST NOT** modify enforcement decisions directly. It provides signals used by the runtime posture synthesis stage of the decision engine.

---

## E.2 Authority Flow Signal

Authority flow represents the rate at which actors attempt to execute transitions toward primitives.

Flow measurement fields:

- primitive
- actor
- transition\_rate
- terminal
- time\_window

Example representation:

```
{
  "primitive": "MONEY",
  "transition_rate": 120,
  "time_window": "60s"
}
```

---

## E.3 Authority Pressure

Authority pressure represents accumulated flow directed toward a primitive or terminal.

Pressure is defined as:

$\text{pressure} = \text{transition\_rate} \times \text{consequence\_weight}$

Where consequence weight reflects primitive severity.

Pressure levels are classified as:

- NORMAL
- ELEVATED
- HIGH
- CRITICAL

When pressure exceeds configured thresholds, a `PRESSURE_SPIKE` observation **MUST** be emitted.

---

#### E.4 Convergence Detection

Authority convergence occurs when multiple authority paths target the same primitive node simultaneously.

Example condition:

```
if distinct_paths(target_primitive) > convergence_threshold:  
    emit STATE_CONVERGENCE_DETECTED
```

Convergence indicates elevated authority risk and potential execution congestion.

---

#### E.5 Reachability Expansion Telemetry

Reachability expansion occurs when new primitives become reachable from an actor.

Detection rule:

```
if reachable_primitives_new > reachable_primitives_previous:  
    emit REACHABILITY_EXPAND
```

Reachability telemetry **MUST** include:

- actor
  - newly reachable primitives
  - expansion source (tool, delegation, integration)
- 

#### E.6 Escalation Chain Telemetry

Escalation telemetry captures primitive transitions across domains.

Example escalation chain:

MODEL → POLICY → AUTHORITY → MONEY

When detected, an `ESCALATION_CROSS` instruction **SHOULD** be emitted.

Telemetry fields:

- primitive\_chain

- escalation\_depth
  - initiating\_actor
- 

## E.7 Containment Zone Telemetry

Telemetry MUST detect containment violations.

Example rule:

```
if actor_zone != permitted_zone:  
    emit ZONE_EXIT_ATTEMPT
```

Zone telemetry fields:

- actor
  - source\_zone
  - attempted\_zone
  - primitive\_target
- 

## E.8 Telemetry Aggregation

Telemetry events are aggregated by the authority control plane.

Aggregations include:

- flow rate by primitive
- pressure trends
- convergence hotspots
- escalation frequency

Aggregated telemetry informs runtime posture synthesis.

---

## E.9 Telemetry Integrity

Telemetry signals MUST be integrity protected to prevent manipulation.

Integrity requirements:

- signed emission
- timestamp validation
- source terminal verification

Telemetry lacking integrity guarantees MUST be treated as degraded evidence.

---

## E.10 Telemetry Output Structure

Example telemetry instruction:

```
{
  "instruction_class": "Observation",
  "opcode": "PRESSURE_SPIKE",
  "primitive": "MONEY",
  "runtime_state": {
    "pressure_level": "HIGH"
  }
}
```

Telemetry instructions are processed by the decision engine during runtime posture synthesis.

## Appendix F — Terminal Enforcement Contracts

This appendix defines the behavioral contract for Authority Terminals. Terminals are distributed enforcement components located near consequence execution points (APIs, payment rails, identity systems, model deployment systems, physical controllers, etc.).

The terminal contract ensures that centralized authority semantics can be safely enforced in distributed environments.

---

### F.1 Terminal Definition

An Authority Terminal is a system component responsible for enforcing AIR control instructions at the point closest to primitive execution.

Terminals are not authority decision-makers. They enforce decisions issued by the Authority Control Plane and may tighten enforcement based on local safety conditions.

---

### F.2 Terminal Classes

AIR defines three terminal capability classes.

#### ENFORCE\_ONLY

Capabilities:

- execute ALLOW or DENY
- block unauthorized requests

**Restrictions:**

- cannot modify enforcement logic
- cannot issue tightening controls

**Example systems:**

- API gateways
- transaction processors

---

**ENFORCE\_AND\_VERIFY****Capabilities:**

- execute control instructions
- verify conditions locally

**Example verification actions:**

- signature verification
- identity attestation
- policy version validation

**Restrictions:**

- cannot broaden authority

---

**ENFORCE\_AND\_TIGHTEN****Capabilities:**

- enforce central controls
- tighten enforcement under local safety conditions

**Example tightening actions:**

- escalate VERIFY → MEDIATION\_REQUIRED
- escalate ALLOW → THROTTLE

**Restrictions:**

- tightening MUST follow AIR safety ordering
-

### F.3 Control Lowering

AIR control instructions MUST be translated into terminal-native actions.

Examples:

| AIR Control        | Terminal Action              |
|--------------------|------------------------------|
| ALLOW              | execute operation            |
| DENY               | block request                |
| THROTTLE           | apply rate limit             |
| BUFFER             | queue request                |
| VERIFY             | require proof or attestation |
| PARTITION          | isolate scope                |
| MEDIATION_REQUIRED | route to adjudication        |
| TERMINATE          | immediately halt operation   |
| ACCEPT_RISK        | execute under risk scope     |

---

### F.4 Monotonic Enforcement Rule

A terminal MUST NOT broaden authority relative to the control plane decision.

Valid transformations:

ALLOW → THROTTLE

ALLOW → VERIFY

ALLOW → DENY

Invalid transformations:

DENY → ALLOW

VERIFY → ALLOW

MEDIATION\_REQUIRED → ALLOW

---

## F.5 Local Safety Elevation

A terminal MAY tighten enforcement when local runtime conditions indicate increased risk.

Trigger conditions may include:

- anomaly signals
- degraded system state
- integrity verification failure
- unsafe terminal environment

When tightening occurs, the terminal MUST emit a LOCAL\_SAFETY\_ELEVATION runtime instruction.

---

## F.6 Terminal Isolation Behavior

When a terminal loses connectivity with the control plane, it enters ISOLATED state.

Isolation rules:

- high-consequence operations MUST be denied
- irreversible transitions MUST be denied
- low-risk operations MAY be buffered

A TERMINAL\_ISOLATED runtime instruction MUST be emitted.

---

## F.7 Enforcement Verification

Terminals MUST confirm enforcement outcomes.

Verification fields include:

- executed\_control
- target\_primitive
- result
- timestamp

Example verification record:

```
{
  "terminal_id": "payments-gateway-3",
  "executed_control": "DENY",
  "primitive": "MONEY",
  "result": "BLOCKED",
  "timestamp": "2026-03-07T10:22:15Z"
```

}

---

## F.8 Enforcement Failure Handling

If a terminal cannot enforce the received control instruction, it **MUST**:

1. deny the request
2. emit an `ENFORCEMENT_FAILURE` telemetry event
3. report the failure to the control plane

Fail-open behavior is **NOT** permitted for high-consequence primitives.

---

## F.9 Terminal Integrity Requirements

Authority terminals **MUST** implement the following protections:

- signed control instructions
- authenticated control plane communication
- tamper-resistant logging

Terminals that fail integrity verification **MUST** enter restricted enforcement mode.

---

## F.10 Terminal Telemetry

Terminals **MUST** emit telemetry for:

- enforcement decisions
- safety elevation events
- boundary violations
- isolation conditions

Telemetry feeds into the Authority Surface Telemetry subsystem.

## Appendix G — Authority Attacks and Failure Modes

This appendix defines the threat and failure model for AIR-enabled authority systems. These threats arise when adversaries manipulate signals, expand reachability, or compromise enforcement components to cause authorized systems to execute harmful consequences.

Authority attacks differ from traditional cyber attacks: instead of breaking systems, they manipulate legitimate authority flows.

---

## G.1 Authority Attack Definition

An Authority Attack occurs when an actor manipulates signals such that authorized systems execute consequences that violate governance intent.

Authority attacks exploit:

- signal trust
- authority path reachability
- escalation chains
- boundary weaknesses
- telemetry blind spots

---

## G.2 Synthetic Insider Abuse

Synthetic insiders are automated actors with legitimate authority.

Examples include:

- AI agents
- automation workflows
- delegated services

Attack pattern:

1. attacker manipulates input signals
2. automation executes authorized actions
3. consequences occur without policy violation at the local level

Mitigation mechanisms:

- escalation boundary enforcement
- containment zones
- authority telemetry

---

## G.3 Signal Manipulation Attacks

Authority systems rely on signals to initiate execution.

Signal manipulation attacks include:

- forged requests
- prompt injection
- malicious API payloads
- delegated workflow abuse

Detection signals:

- anomaly elevation
- boundary violations
- unexpected escalation chains

---

## G.4 Escalation Chain Exploitation

Attackers may intentionally trigger escalation sequences to gain access to higher-consequence primitives.

Example chain:

IDENTITY → POLICY → AUTHORITY → MONEY

Attack strategy:

1. gain identity privileges
2. modify policy controls
3. obtain authority delegation
4. execute financial transfer

Mitigation requires explicit escalation boundaries between primitives.

---

## G.5 Authority Surface Expansion Attacks

Authority surface expansion occurs when reachable primitives increase.

Attack methods include:

- adding integrations
- creating new delegations
- granting tool access

Expansion attacks are dangerous because they may appear legitimate.

Detection signal:

REACHABILITY\_EXPAND

---

## G.6 Boundary Bypass

Boundary bypass occurs when actors reach primitives without crossing required governance boundaries.

Causes include:

- misconfigured architecture
- unintended integration paths
- direct API access

AIR mitigation:

- path discovery validation
- boundary enforcement at terminals

---

## G.7 Authority Convergence Attacks

Convergence attacks overwhelm primitives by directing many authority paths toward the same consequence node.

Examples:

- coordinated transaction bursts
- simultaneous policy modification attempts

Telemetry signal:

STATE\_CONVERGENCE\_DETECTED

Mitigation:

- throttling
- buffering
- partitioning

---

## G.8 Telemetry Deception

Attackers may attempt to manipulate telemetry signals.

Examples:

- forged telemetry events
- suppressed anomaly signals
- manipulated pressure indicators

Mitigation mechanisms:

- signed telemetry
  - cross-terminal correlation
  - control plane verification
- 

## G.9 Terminal Compromise

A compromised terminal may attempt to bypass enforcement decisions.

Failure scenarios:

- unauthorized ALLOW execution
- suppression of DENY
- falsified enforcement logs

Required mitigation:

- integrity verification
  - enforcement audits
  - replay verification
- 

## G.10 Control Plane Failure

Control plane failures may arise from:

- synchronization loss
- corrupted semantic epochs
- decision engine faults

Mitigation rules:

- terminals enter restricted mode
  - high-consequence primitives default to DENY
- 

## G.11 Authority Collapse

Authority collapse occurs when authority boundaries erode and many primitives become reachable without mediation.

Indicators:

- rapid reachability expansion

- frequent escalation chains
- loss of boundary enforcement

Mitigation:

- restore boundary architecture
- isolate affected zones

---

## G.12 Failure Detection Requirements

AIR systems MUST detect and report:

- boundary violations
- escalation chains
- convergence events
- terminal isolation
- reachability expansion

Failure detection signals MUST be recorded for replay and audit.

## Appendix H — Example End-to-End Execution Trace

This appendix demonstrates a complete AIR evaluation sequence for a realistic authority scenario. The goal is to illustrate how AIR-S semantics, AIR-R runtime instructions, and distributed enforcement interact during execution.

Scenario: An AI agent attempts to initiate a financial transfer through an automated workflow.

Primitive escalation chain:

MODEL → POLICY → AUTHORITY → MONEY

---

### H.1 AIR-S Structural Context

Relevant AIR-S nodes:

- Actor: ai\_agent\_7 (type: AI\_AGENT)
- Service: workflow\_engine
- Policy Node: payment\_policy\_v3
- Authority Node: finance\_delegation\_scope
- Primitive: MONEY

Relevant edges:

- ai\_agent\_7 CAN\_TOUCH workflow\_engine

- workflow\_engine CAN\_TRANSITION payment\_policy\_v3
- payment\_policy\_v3 CAN\_TRANSITION finance\_delegation\_scope
- finance\_delegation\_scope REACHES MONEY

Boundary conditions:

- boundary\_approval\_required
- boundary\_transfer\_threshold

Terminal:

- payments\_gateway\_terminal

---

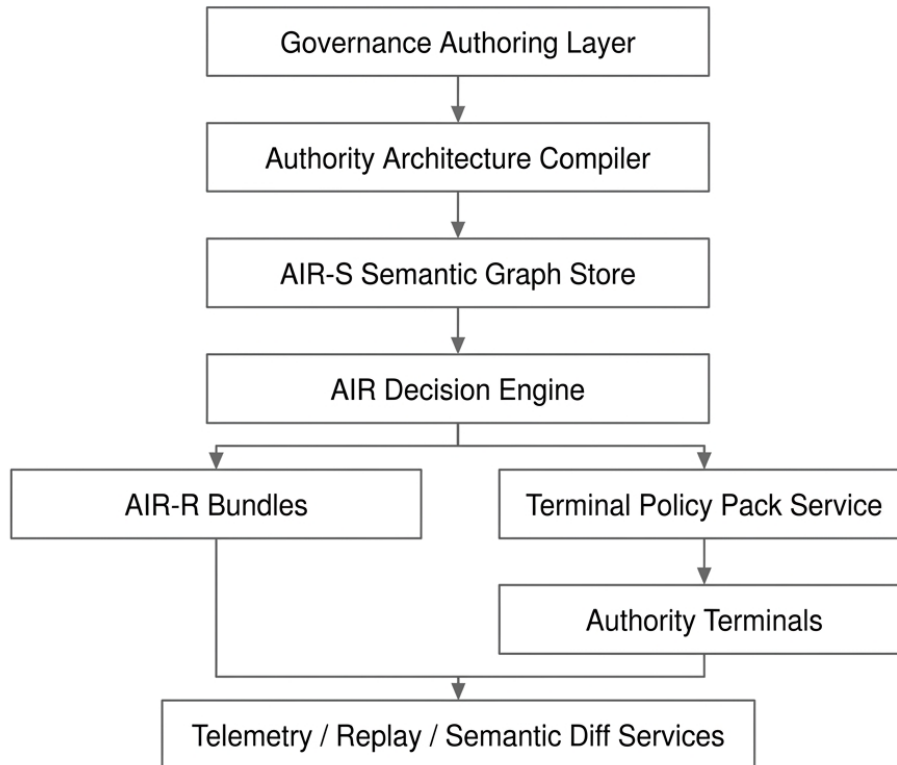
## H.2 Incoming AIR-R Observation

Initial event generated by the workflow system:

```
{
  "instruction_class": "Observation",
  "opcode": "ACTION_REQUEST",
  "actor": {
    "actor_id": "ai_agent_7",
    "actor_type": "AI_AGENT"
  },
  "action": {
    "action_type": "transfer_funds",
    "amount": 50000
  },
  "primitive": {
    "primitive_type": "MONEY"
  }
}
```

}

## Logical AIR System Architecture




---

### H.3 Primitive Binding

Primitive inference confirms the target primitive as MONEY.

Derived primitive chain:

MODEL → POLICY → AUTHORITY → MONEY

Escalation depth = 3

---

### H.4 Authority Path Discovery

Discovered authority path:

```

ai_agent_7
  → workflow_engine
  → payment_policy_v3
  
```

→ finance\_delegation\_scope  
→ MONEY

Path classification: DERIVED

Delegation depth: 2

Boundaries encountered:

- boundary\_transfer\_threshold

---

## H.5 Boundary Evaluation

Boundary rule:

Transfers above 10,000 require mediation.

Evaluation result:

amount = 50000  
threshold = 10000

boundary\_transfer\_threshold = violated

Boundary result: mediation required.

---

## H.6 Runtime Posture

Runtime posture signals:

- sync\_state = FRESH
- anomaly = LOW
- pressure = NORMAL

Runtime confidence: SUFFICIENT

---

## H.7 Admissible Control Set

Given the boundary violation:

Admissible controls:

{DENY, MEDIATION\_REQUIRED}

---

## H.8 Control Selection

Safety ordering selects:

MEDIATION\_REQUIRED

Reason: boundary threshold exceeded.

---

## H.9 AIR-R Control Bundle

The decision engine emits the following instructions:

```
[
  { "opcode": "PRIMITIVE_TOUCH" },
  { "opcode": "BOUNDARY_CHECK" },
  { "opcode": "ESCALATION_CROSS" },
  { "opcode": "MEDIATION_REQUIRED" }
]
```

---

## H.10 Terminal Enforcement

The payments gateway terminal receives the control instruction.

Terminal action:

- block transaction
- route request to human finance approver

Verification record:

```
{
  "terminal_id": "payments_gateway_terminal",
  "executed_control": "MEDIATION_REQUIRED",
  "primitive": "MONEY",
  "result": "AWAITING_APPROVAL"
}
```

---

## H.11 Explanation Artifact

Generated reason graph includes:

Facts:

- actor = ai\_agent\_7
- primitive = MONEY
- transfer\_amount = 50000

Inference:

- escalation\_chain\_detected

Boundary outcome:

- transfer\_threshold\_exceeded

Admissibility:

- allow\_removed

Decision:

- mediation\_required

---

## H.12 Replay

Replay inputs:

- semantic\_epoch
- original request
- runtime context

Replay MUST reproduce the decision:

MEDIATION\_REQUIRED

If replay produces a different decision, the system MUST identify semantic drift or evidence mismatch.

## Appendix I — AIR Formal Model

This appendix defines the formal mathematical model underlying the Authority Intermediate Representation (AIR). The goal is to make the semantics of AIR precise and analyzable while remaining consistent with the Authority Engineering framework.

The formal model defines:

- the AIR-S semantic graph
- authority paths
- boundary constraints

- admissible control computation
- decision selection

## I.1 AIR-S Semantic Graph

AIR-S is modeled as a typed labeled multigraph:

$$G = (V, E, \tau_V, \tau_E, \lambda_V, \lambda_E)$$

Where:

- $V$  = set of nodes
- $E$  = set of edges
- $\tau_V : V \rightarrow \text{NodeTypes}$
- $\tau_E : E \rightarrow \text{EdgeTypes}$
- $\lambda_V : V \rightarrow \text{attribute maps}$
- $\lambda_E : E \rightarrow \text{attribute maps}$

NodeTypes include:

Primitive, Actor, Zone, Terminal, Boundary, Path, DecisionPolicy

EdgeTypes include:

CAN\_TOUCH, CAN\_TRANSITION, DELEGATES\_TO, REACHES, CROSSES, BOUNDED\_BY, ENFORCED\_AT

The graph encodes all structural authority relationships in the system.

## I.2 Primitive Set

AIR assumes primitive closure.

Let:

$P = \{\text{Money, Identity, Policy, Authority, Model, Physical, Contract}\}$

Every consequence-bearing transition must terminate in at least one primitive  $p \in P$ .

A primitive transition is represented as:

$(p, \text{state\_before}, \text{state\_after})$

### I.3 Authority Paths

An authority path is a sequence of graph edges connecting an actor to a primitive:

$$\pi = (v_0, e_1, v_1, e_2, v_2, \dots, e_n, v_n)$$

Where:

$$v_0 = \text{actor } v_n = \text{primitive}$$

Each edge must satisfy traversal rules defined by AIR-S.

The set of all reachable paths from actor  $a$  to primitive  $p$  is:

$$\text{Paths}(a, p)$$

### I.4 Path Classification Function

Each path is assigned a classification:

$$\text{Classify} : \text{Paths} \rightarrow \{\text{DECLARED}, \text{DERIVED}, \text{EMERGENT}, \text{ILLEGITIMATE}\}$$

Definitions:

DECLARED  $\pi$  is explicitly represented in the AIR-S path graph.

DERIVED  $\pi$  is logically implied by delegation or transition edges.

EMERGENT  $\pi$  arises from runtime integration or reachability expansion.

ILLEGITIMATE  $\pi$  violates architectural constraints.

### I.5 Boundary Constraints

Boundaries are constraints attached to edges or nodes.

Each boundary is a predicate:

$$B : \text{Context} \rightarrow \{\text{true}, \text{false}, \text{pending}\}$$

Examples:

- approval required
- threshold limits

- mediation rules

A path is admissible only if all required boundaries evaluate to true or pending mediation.

## I.6 Runtime Context

The runtime context is defined as:

$$C = (\text{sync\_state}, \text{anomaly\_level}, \text{pressure\_level}, \text{terminal\_health})$$

Where each element represents observed system state.

Runtime confidence is derived as a function:

$$\text{Confidence}(C) \in \{\text{SUFFICIENT}, \text{DEGRADED}, \text{INSUFFICIENT}\}$$

## I.7 Admissible Control Set

Let Controls be the set of possible enforcement actions:

$$\text{Controls} = \{\text{ALLOW}, \text{THROTTLE}, \text{BUFFER}, \text{VERIFY}, \text{MEDIATION\_REQUIRED}, \text{DENY}, \text{TERMINATE}\}$$

The admissible control set is defined as:

$$A = \text{Admissible}(a, p, \pi, B, C)$$

Where:

- $a$  = actor
- $p$  = primitive
- $\pi$  = authority path
- $B$  = boundary results
- $C$  = runtime context

The admissible function removes controls that violate AIR laws.

Example constraint:

if  $\text{Classify}(\pi) = \text{ILLEGITIMATE}$

then

$$A = \{\text{DENY}, \text{TERMINATE}\}$$

## I.8 Safety Ordering

AIR defines a partial ordering over controls:

ALLOW < THROTTLE < BUFFER < VERIFY < MEDIATION\_REQUIRED < DENY < TERMINATE

This ordering defines increasing enforcement strictness.

---

## I.9 Decision Selection

The decision function selects the least-dangerous admissible control:

Decision = min(A)

Where min is defined with respect to the AIR safety ordering.

---

## I.10 Execution Semantics

Given an event  $e$  and semantic graph  $G$ , the decision process is:

1. determine primitive  $p$
2. compute  $\text{Paths}(\text{actor}, p)$
3. evaluate boundary predicates
4. derive runtime context
5. compute admissible controls
6. select minimal admissible control

The resulting control instruction is emitted as an AIR-R instruction.

---

## I.11 Determinism Requirement

For a fixed semantic epoch and evidence set:

Decide( $e, G, C$ )

must produce a deterministic decision.

If nondeterminism occurs, the system must record the sources of variation.

---

## I.12 Relation to Authority Surface

The authority surface can be defined formally as:

$$AS = \{ (a, p) \mid \exists \pi \in \text{Paths}(a, p) \}$$

Where:

- $a$  = actor
- $p$  = primitive

The authority surface represents all reachable consequence transitions within the system.

AIR evaluation determines whether a specific transition on this surface is permitted.

## Appendix J — AIR Reference Architecture

This appendix defines the reference implementation architecture for AIR as an operational platform. The goal is to translate the AIR specification into a buildable system design that preserves centralized authority semantics with distributed consequence enforcement.

The AIR Reference Architecture defines the major system components, data flows, trust boundaries, and deployment patterns required to implement AIR in production environments.

---

### J.1 Architectural Objective

The AIR platform exists to compile governance intent into executable authority semantics and enforce those semantics near consequence execution points.

The architectural objective is:

- centralized semantic control
- distributed primitive-adjacent enforcement
- replayable and explainable decision making
- continuous authority surface observation

---

### J.2 Core Architectural Principle

AIR is built around a split-control model:

- the Authority Control Plane owns semantic truth
- Authority Terminals own consequence-near enforcement

This allows the system to preserve a single semantic model while remaining resilient in distributed environments.

---

### J.3 Reference Architecture Components

The AIR Reference Architecture contains eight primary components.

#### 1. Governance Authoring Layer

Purpose:

- define governance intent
- specify mediation requirements
- assign stop rights
- define risk acceptance authorities

Outputs:

- governance policies
  - authority design inputs
- 

#### 2. Authority Architecture Compiler

Purpose:

- compile governance and architecture definitions into AIR-S
- validate primitive mappings
- validate boundary completeness
- produce semantic epochs

Outputs:

- AIR-S graph
  - module hashes
  - semantic epoch
  - terminal policy projections
- 

#### 3. AIR Semantic Graph Store

Purpose:

- persist AIR-S modules
- serve graph queries to the decision engine

- version semantic epochs
- support replay and diff

Stored artifacts:

- primitive registry
  - path graph
  - boundary graph
  - decision model
  - terminal model
- 

#### 4. AIR Decision Engine

Purpose:

- evaluate live requests
- bind primitive truth
- discover authority paths
- evaluate boundaries
- compute admissible controls
- emit AIR-R bundles

Outputs:

- control decisions
  - explanation artifacts
  - replayable bundles
- 

#### 5. AIR Telemetry and Surface Analytics Layer

Purpose:

- ingest runtime telemetry
- compute authority pressure
- detect convergence
- detect surface expansion
- feed runtime posture synthesis

Outputs:

- telemetry aggregates
  - posture signals
  - anomaly indicators
-

## 6. Terminal Policy Pack Service

Purpose:

- project AIR-S subsets to terminals
- package enforcement rules by primitive adjacency
- manage terminal capability bindings
- distribute signed policy packs

Outputs:

- terminal-specific semantic projections
  - signed local enforcement packs
- 

## 7. Authority Terminals

Purpose:

- enforce AIR controls near primitive execution
- verify signed controls
- locally tighten under bounded rules
- emit terminal outcome telemetry

Examples:

- payment gateway terminal
  - identity mutation terminal
  - model deployment terminal
  - robotic safety terminal
  - contract execution terminal
- 

## 8. Replay, Audit, and Semantic Diff Service

Purpose:

- replay decisions under prior epochs
- compare expected vs actual terminal outcomes
- detect semantic drift
- support audit and incident response

Outputs:

- replay reports
- deviation findings
- cross-epoch control diffs

---

## J.4 Logical System Diagram

### Logical AIR System Architecture

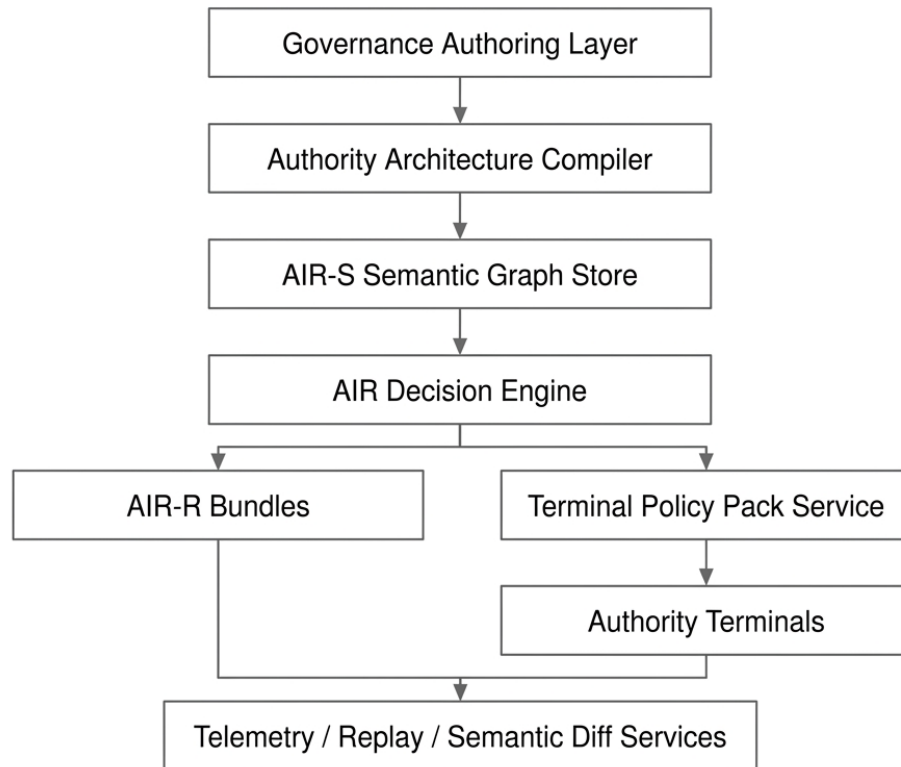


Figure J4. Logical AIR System Architecture

The AIR system separates governance authoring, authority compilation, runtime decision evaluation, distributed enforcement, and telemetry-based observability.

---

## J.5 Data Flow

The reference runtime data flow proceeds as follows:

## AIR Runtime Data Flow

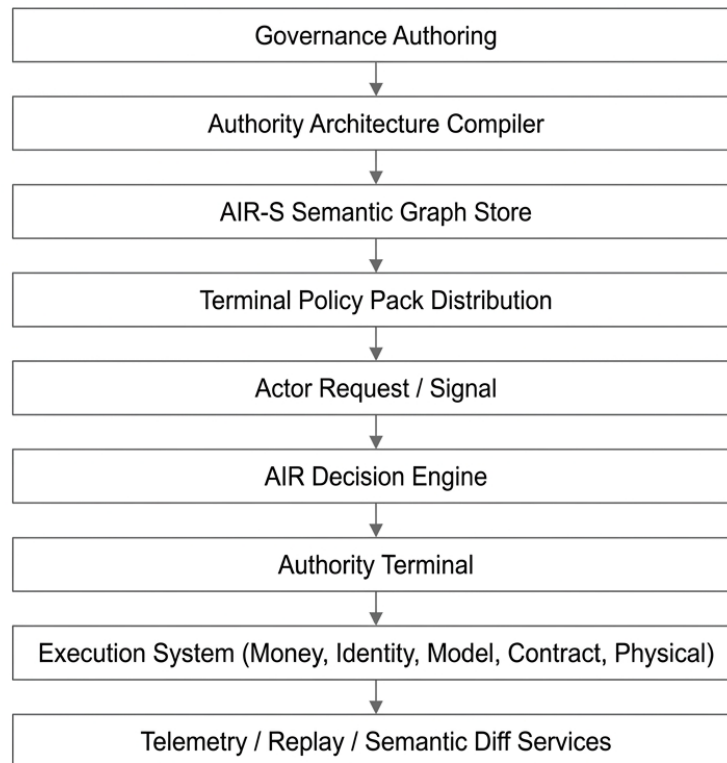


Figure J5 – AIR Runtime Data Flow

### Step 1: Authoring

Governance and architecture teams define authority rules.

### Step 2: Compilation

The compiler validates and emits AIR-S.

### Step 3: Distribution

AIR-S is stored centrally. Terminal projections are distributed as policy packs.

### Step 4: Runtime Request

An actor emits a signal or request toward a consequence-bearing system.

### Step 5: Decision Evaluation

The AIR Decision Engine queries AIR-S, computes admissible controls, and emits AIR-R instructions.

### Step 6: Terminal Enforcement

The relevant terminal receives the control and enforces it.

### Step 7: Telemetry Return

Terminals emit results and telemetry back to the control plane.

### Step 8: Replay and Audit

All decisions remain replayable under their semantic epoch.

---

## J.6 Deployment Topology

The AIR platform is best deployed in four layers.

### Layer A — Authoring and Compilation

Centralized, low-frequency change domain.

### Layer B — Semantic Core Services

Centralized, high-integrity services:

- graph store
- compiler outputs
- decision engine

### Layer C — Telemetry and Audit Services

Semi-centralized analytics and replay services.

### Layer D — Distributed Terminals

Primitive-adjacent enforcement nodes deployed close to execution systems.

This layered model preserves the distinction between semantic truth and consequence enforcement.

## J.7 Trust Boundaries

The architecture contains four key trust boundaries.

### Boundary 1: Governance to Compiler

Ensures policy intent is translated without semantic loss.

### Boundary 2: Compiler to AIR-S Store

Ensures semantic graph integrity and epoch consistency.

### Boundary 3: Decision Engine to Terminals

Ensures control instructions are signed, authentic, and non-broadenable.

### Boundary 4: Terminal to Telemetry Plane

Ensures enforcement results and local safety elevations are trustworthy.

These trust boundaries MUST be integrity-protected.

---

## J.8 Reference Storage Model

A production AIR platform should store at least the following artifacts:

### Static Stores

- AIR-S graph modules
- semantic epoch history
- terminal capability profiles
- projection packs

### Runtime Stores

- AIR-R instruction bundles
- explanation objects
- telemetry streams
- terminal enforcement receipts

### Audit Stores

- replay evidence
- deviation findings
- semantic diff results

This separation supports performance and auditability.

---

## J.9 Authority Compiler Outputs

The compiler should produce the following deployable artifacts:

1. Canonical AIR-S graph
2. Module hashes and semantic epoch
3. Decision admissibility tables
4. Terminal projection packs
5. Replay compatibility metadata

These artifacts together define the active semantic release.

---

## J.10 Decision Engine Internal Services

A reference AIR decision engine should itself contain specialized services.

### Primitive Binding Service

Maps runtime actions to canonical primitives.

### Path Discovery Service

Traverses AIR-S to locate candidate authority paths.

### Boundary Evaluation Service

Evaluates node, edge, and path-level constraints.

### Runtime Posture Service

Synthesizes sync, anomaly, pressure, and terminal health.

### Admissibility Evaluator

Computes the legal control set.

### Control Selector

Chooses the least-dangerous admissible control.

### Explanation Builder

Constructs reason graphs.

### AIR-R Bundle Generator

Emits runtime instruction bundles.

These can be separate modules or one service internally partitioned.

---

## J.11 Terminal SDK Concept

A practical AIR implementation should provide a Terminal SDK.

The SDK should provide:

- control signature verification
- policy pack loading
- local monotonic tightening logic
- enforcement receipt generation
- telemetry emission
- replay identifiers and bundle lineage tracking

This reduces implementation variance across terminals.

---

## J.12 Terminal Modes

The reference architecture assumes terminals support explicit runtime modes.

- READY
- EVALUATING
- BUFFERING
- THROTTLING
- MEDIATED
- PARTITIONED
- ISOLATED
- TERMINATED

These modes SHOULD be reflected in terminal telemetry and audit logs.

---

## J.13 Semantic Epoch Management

Semantic epochs are central to AIR correctness.

The architecture MUST support:

- active semantic epoch
- staged next epoch
- prior epoch retention for replay
- epoch compatibility checks at terminals

A semantic epoch rollout should include:

1. compile
2. validate
3. sign
4. distribute
5. activate
6. monitor

---

## J.14 Resilience Model

AIR must remain safe under degraded conditions.

### Control Plane Degradation

If the central plane is unavailable, terminals enter bounded enforcement mode.

### Graph Store Degradation

The decision engine may use cached semantic projections within freshness bounds.

### Telemetry Degradation

Decision quality may degrade, but boundary rules remain authoritative.

### Terminal Isolation

High-consequence operations default to deny or mediation.

The architecture is therefore fail-safe, not fail-open, for high-consequence primitives.

---

## J.15 Security Controls

A production AIR platform should include at least these protections:

- signed AIR-S releases
- signed AIR-R control instructions

- mutual authentication between control plane and terminals
- tamper-evident logging
- bundle hash chaining
- telemetry integrity verification
- least-privilege service identities

These are not optional in high-consequence environments.

---

## J.16 Observability Model

The architecture should expose observability at three levels.

### Semantic Observability

- current semantic epoch
- policy pack version
- active path classifications

### Runtime Observability

- decision counts by primitive
- mediation frequency
- deny reasons
- local safety elevations

### Surface Observability

- reachability expansions
- pressure spikes
- convergence hotspots
- zone exits

This gives AIR both operational and structural visibility.

---

## J.17 Integration Patterns

AIR can integrate with different execution environments.

### API Gateways

Enforce request-level controls for Money, Identity, and Policy primitives.

## Workflow Engines

Inject mediation, buffering, and verification steps into automated processes.

## IAM / PAM Systems

Control identity binding and authority assignment.

## Model Platforms

Control model promotion and model-to-policy transitions.

## Robotics / OT Controllers

Enforce physical safety controls and safe-mode constraints.

## Contract Systems

Block or mediate legal commitments.

These are target profiles, not separate AIR variants.

---

## J.18 Reference Product Surfaces

If AIR were implemented as a platform, the main product surfaces would likely be:

- Governance authoring console
- AIR-S graph explorer
- Decision engine API
- Terminal policy pack registry
- Telemetry and authority surface dashboard
- Replay and semantic diff console

These surfaces align to the architecture rather than redefining it.

---

## J.19 Minimal Deployable AIR Platform

A minimal implementation can be built with five deployable components:

1. AIR compiler
2. AIR-S graph store
3. AIR decision API
4. one terminal SDK and terminal implementation
5. replay / audit service

This is sufficient to prove the architecture in one domain such as payments or identity.

---

## J.20 Recommended Implementation Sequence

A practical build sequence is:

### Phase 1

- primitive registry
- path graph
- boundary graph
- decision engine core
- one terminal class

### Phase 2

- telemetry aggregation
- explanation service
- replay service
- semantic diff

### Phase 3

- multi-terminal support
- local tightening support
- policy pack distribution
- cross-zone deployment

### Phase 4

- full authority surface analytics
- adversarial testing
- epoch impact simulation

This sequence reduces implementation risk while preserving the AIR architecture.

---

## J.21 Canonical Reference Architecture Principle

The canonical AIR principle is:

Centralize semantic authority. Distribute consequence enforcement. Preserve replayable meaning end-to-end.

That sentence captures the platform architecture in one line.

