ARETABA: A Formal Execution-Time Authority Framework for Constructing and Enforcing Governed Irreversible Actions in Agentic AI Systems

Masayuki Otani
Architectural Governance
United Kingdom
www.architecturalgovernance.com
info@architecturalgovernance.com

02 March 2026

Abstract

As agentic AI systems move from advisory assistance to execution-bearing architectures, governance must remain enforceable at the moment a system-controlled action becomes irreversible. OTANIS defines a boundary-evaluated permission architecture and ISDAIRE defines ex-ante admissibility, but neither by itself answers the narrower construction question of how runtime authority is systematically built, bound, and resolved for a governed action instance. This paper isolates and formalises ARETABA as the execution-time authority framework that answers that question.

ARETABA is presented as both a minimal runtime control surface and a construction discipline for authority objects at the earliest governed irreversible execution boundary $T_e^*(a)$. The framework specifies how authority scope, boundary definition, lifecycle validity, revocation, refusal, escalation, traceability, accountability, and boundary-evaluated admissibility are derived from governed ex-ante artefacts, versioned boundary registries, and declared dependency scope. A class-level synthesis function is introduced to derive authority templates from ISDAIRE-governed action classes, together with a runtime resolution function that instantiates boundary-resolved authority objects for concrete action instances.

The paper formalises completeness, consistency, and executability criteria for ARETABA templates and shows how these criteria support the boundary-evaluated admissibility predicate already used in OTANIS. The framework is intentionally fail-closed, replay-oriented, and bounded to action classes whose candidate boundaries, ordering semantics, dependency scope, and controlled mediated commit paths can be declared ex ante and checked at runtime. It does not infer omitted hazards, omitted dependencies, or semantically incomplete authority content after deployment.

A detailed insurance payout example shows how ARETABA turns an admissible action class into a runtime-enforceable authority structure and where failure remains a specification problem rather than an execution-boundary enforcement problem. The paper positions ARETABA as the execution-time authority discipline underpinning OTANIS, complementing ISDAIRE at the design layer and GAG/MGAG at the compositional layer.

Keywords: Agentic AI, execution-time authority, architectural governance, admissibility, irreversibility, traceability, auditability.

1 Introduction

The move from AI systems that recommend to AI systems that commit creates a narrower but much harder governance problem. The relevant question is no longer only whether a model output looks plausible, aligned, or low-risk. The relevant question is whether a system can demonstrate and enforce legitimate authority at the point where it crosses into irreversible external effect.

That requirement is structurally aligned with the principle of complete mediation, namely that every protected operation must be checked at the time it occurs [1]. In execution-bearing AI systems, the protected operation is the irreversible action itself. Prior work formalised this point by treating governance as an executable control structure operating at the point of irreversibility rather than as a policy layer or a post-hoc explanation [2]. OTANIS develops that line further by specifying an integrated architecture for declared-boundary execution governance across multiple candidate boundaries and multiple governance layers [3].

Within OTANIS, one predicate is especially important:

$$\mathsf{Permit}(a, \beta_e(a), x_e(a)) \tag{1}$$

where a is an action instance, $x_e(a)$ is the execution-boundary snapshot state, and $\beta_e(a)$ is the resolved authority object at the bound earliest governed execution boundary $T_e^*(a)$. OTANIS defines the conditions under which this permit relation may hold. What it does not isolate as a standalone framework is the narrower construction question:

> How is the runtime authority object $\beta_e(a)$ systematically constructed, version-bound, dependency-complete, and made resolvable at $T_e^*(a)$?

That question matters because execution-time governance does not fail only when the permit predicate is ignored. It also fails when the authority object was never constructed rigorously in the first place. A system may appear to be checking permissions while silently relying on incomplete scope, undeclared revocation sources, ambiguous boundary definitions, missing trace commitments, or non-attributable authority surfaces. In that case the permit check is formally present but materially insufficient.

This paper isolates ARETABA as the execution-time authority framework that closes that gap. ARETABA is not introduced here as a new top-level alternative to OTANIS. It is the execution-time authority discipline inside OTANIS. Its role is to specify how the runtime authority object is built, what fields it must contain, how those fields are derived from governed ex-ante artefacts, what completeness and consistency conditions apply, and how the resulting object is resolved and enforced at $T_e^*(a)$.

The contribution claimed here is therefore narrower than OTANIS and intentionally so. The paper does not claim to solve semantic adequacy in the abstract. It does not discover omitted hazards, infer undeclared dependencies, or reconstruct true boundary semantics from arbitrary live systems after deployment. Instead, it makes precise what a governable execution-time authority object must look like and how a compliant system must construct and enforce it. This makes the framework relevant for design, review, procurement qualification, and forensic audit in precisely those settings where execution-bearing action classes must be governed rather than merely supervised [4--6].

2 What ARETABA Contributes as an Architectural Framework

ARETABA contributes two things that are often conflated but should be kept distinct.

First, it contributes a *minimal runtime control surface*. In OTANIS terms, this is the boundary-evaluated conjunction that requires:

- scope-valid authority,
- applicable boundary constraints,
- traceability,
- accountability,
- dependency conformance, and
- state freshness

before an irreversible action may be committed.

Second, it contributes a *construction discipline*. This is the missing step between ex-ante admissibility and runtime permit evaluation. It defines how authority scope, boundary definition, lifecycle validity, revocation, refusal, escalation, traceability, and accountability are systematically synthesised from governed artefacts rather than assumed to exist implicitly.

This distinction matters. The runtime predicate by itself can only say whether a given β is admissible at a boundary. It cannot say whether β was constructed from a complete dependency set, whether revocation sources were declared properly, whether accountability terminates at a legitimate authority surface, or whether the trace schema is sufficient for deterministic replay. ARETABA therefore fills the construction gap between ISDAIRE and OTANIS.

Read architecturally, the division of labour is as follows:

- **ISDAIRE** specifies whether an action class is admissible for governed execution in principle.
- **ARETABA** specifies how execution-time authority is constructed and made enforceable in fact.
- **OTANIS** integrates those elements into a boundary-enforced permission architecture.
- **GAG/MGAG** preserve legitimacy across composed and multi-layer systems.

ARETABA is therefore not a checklist and not a mere labelling scheme. It is the execution-time authority framework that makes runtime governance structurally possible.

3 Scope and Non-Claims

3.1 Scope

This paper applies to execution-bearing action classes in agentic AI systems when the following conditions hold.

1. The action class can produce an irreversible external effect, such as financial settlement, legal commitment, access-control change, physical actuation, or irreversible disclosure.
2. The candidate execution boundaries for that action class can be declared ex ante as a finite set.
3. The earliest governed irreversible execution boundary can be bound deterministically from that set under declared ordering semantics and declared runtime inputs.

4. The final governed commit path is controlled, interceptable, mediated, and non-bypass.

This is intentionally restrictive. The paper is not aimed at offline analytics, advisory copilots, batch inference, or workflows with no meaningful governed execution boundary. It addresses action classes for which execution-time authority must be demonstrable, enforceable, and auditable under runtime pressure.

3.2 Non-Claims

ARETABA does not guarantee correctness, domain adequacy, safety in the broad sense, legal compliance, or organisational acceptability. It does not discover omitted boundaries at runtime, infer undeclared dependencies, or repair semantically insufficient action-class specifications after deployment.

Its guarantees are narrower and more defensible. Where the action class is admissible under ISDAIRE, where the boundary is declared and governable, where the authority template is complete and consistent, and where runtime state remains fresh and conformant, ARETABA makes it possible to construct, resolve, and enforce a runtime authority object at $T_e^*(a)$. Where those conditions fail, it requires refusal or escalation rather than speculative continuation.

A further non-claim is critical. $\mathsf{CompleteARETABA}(\alpha) = 1$ and $\mathsf{ConsistentARETABA}(\alpha) = 1$ do *not* prove that the declared authority content is semantically sufficient for every real-world hazard. They prove only that the declared execution-time authority structure is internally complete and consistent relative to the governed artefacts. Semantic inadequacy remains a design-time governance failure, not a runtime enforcement failure. This claim boundary is deliberate and necessary if the framework is to remain defensible.

4 Architectural Premise

ARETABA is founded on the following premise.

P1. If runtime authority for an action instance cannot be constructed, resolved, and enforced at the earliest governed irreversible execution boundary $T_e^*(a)$, governance did not exist for that action path.

This premise is a direct execution-time extension of complete mediation [1]. In the present context, the question is not whether some policy document exists or whether some upstream signal suggested permission. The question is whether the authority object that justifies the irreversible action exists as an executable, version-bound, attributable, traceable, and dependency-complete object at the governed boundary.

Stated more practically, if a system cannot answer all of the following questions at $T_e^*(a)$, it is not governance-complete for that action path:

- Who is the legitimate authority source for this action instance?
- Is the action class within authorised scope?
- Is the current boundary the correct governed boundary for this path?
- Are lifecycle validity and revocation still current under fresh state?
- Is refusal or escalation defined if the answer is uncertain?
- Is the decision attributable and replayable after the fact?

ARETABA is the formal framework proposed in this paper for constructing that answer.

5 Formal Model and Notation

This paper is intentionally aligned to the OTANIS notation and claim surface [3]. The aim is not to create a parallel notation system. It is to isolate the ARETABA layer while remaining mathematically consistent with the integrated OTANIS architecture.

5.1 System, Action Classes, and Boundary Model

Let an agentic system be represented as a directed graph

$$S = (V, E) \tag{2}$$

where V denotes agents, tools, services, and control components, and E denotes invocation or data-flow edges.

Let $\mathcal{I}$ be the set of action instances the system may attempt, and let $\mathcal{A}_c$ be the finite set of declared action classes. Every action instance $a \in \mathcal{I}$ must be mapped to exactly one declared class $\alpha(a) \in \mathcal{A}_c$. Unclassified actions are non-admissible.

Let T be a set of execution events equipped with a partial order $\prec$ representing causal precedence [7].

Define the declared finite candidate set function

$$\mathsf{Cand}_\alpha : \mathcal{A}_c \to \mathcal{P}_{fin}(T) \tag{3}$$

and for an action instance a,

$$\mathsf{Cand}(a) := \mathsf{Cand}_\alpha(\alpha(a)). \tag{4}$$

Define the deterministic boundary-selection function

$$\mathsf{SelectEarliest} : \mathcal{P}_{fin}(T) \times X \to T \tag{5}$$

where X is the declared domain of runtime selection inputs used for boundary binding.

The bound earliest governed irreversible execution boundary is

$$T_e^*(a) := \mathsf{SelectEarliest}(\mathsf{Cand}(a), x_{sel}). \tag{6}$$

As in OTANIS, ‘‘earliest’’ means earliest under the declared ordering semantics for the action class, not a universal metaphysical notion of temporal precedence across arbitrary infrastructures [3].

5.2 ISDAIRE Preconditions

Because ARETABA constructs runtime authority from governed ex-ante artefacts, the relevant ISDAIRE predicates must be restated here. Let the seven ISDAIRE governance predicates over action class α be:

- $I(\alpha)$ for Intent,
- $S(\alpha)$ for Scope,
- $DS(\alpha)$ for Domain Separation,

- $AS(\alpha)$ for Authority Source,
- $IA(\alpha)$ for Irreversibility Awareness,
- $RF(\alpha)$ for Risk Framing,
- $EB(\alpha)$ for Execution Boundary.

Define

$$\begin{aligned} ISDAIRE(\alpha) := I(\alpha) \wedge S(\alpha) \wedge DS(\alpha) \wedge AS(\alpha) \\ \wedge IA(\alpha) \wedge RF(\alpha) \wedge EB(\alpha). \end{aligned} \tag{7}$$

Define the ex-ante admissibility predicate

$$\begin{aligned} \mathsf{AdmissibleISDAIRE}(\alpha) = 1 \Leftrightarrow I(\alpha) = 1 \wedge S(\alpha) = 1 \wedge DS(\alpha) = 1 \\ \wedge AS(\alpha) = 1 \wedge IA(\alpha) = 1 \wedge RF(\alpha) = 1 \\ \wedge EB(\alpha) = 1 \wedge Consistent(EB(\alpha), IA(\alpha)) = 1. \end{aligned} \tag{8}$$

ARETABA operates only after these ex-ante admissibility conditions have been defined for the action class. It does not replace them. In this paper, $\mathsf{AdmissibleISDAIRE}(\alpha) = 1$ is treated as a structural upstream precondition, not as a claim that the action class is semantically sufficient, safe, legally adequate, or operationally fit for autonomous execution in the broader sense. If the ISDAIRE artefacts are semantically weak, incomplete, or boundary-misclassified, ARETABA may still construct and enforce the resulting runtime authority structure faithfully. In that case the failure remains a design-time semantic-adequacy or boundary-discovery failure rather than an execution-time authority-enforcement failure.

5.3 Boundary Registry and Dependency Scope

Let the versioned Boundary Registry be

$$R = \{(\alpha, \mathsf{Cand}_\alpha(\alpha), \mathsf{SelectEarliest}_\alpha, Order(\alpha), \mathsf{Dep}(\alpha), IA(\alpha), v)\} \tag{9}$$

where $v \in \mathbb{N}$ is the registry version and $\mathsf{Dep}(\alpha) \subseteq \mathrm{Keys}(x)$ is the declared dependency scope for action class α.

For a concrete action instance a, let $\mathsf{Dep}^{\mathrm{inst}}(a)$ denote the deterministic realised instantiation of $\mathsf{Dep}(\alpha(a))$ under any declared parameterisation rules, exactly as in OTANIS [3].

Define the observed dependency set $\mathsf{ObservedDep}(a)$ as the set of state keys actually read during execution-boundary governance evaluation at $T_e^*(a)$. Then

$$\mathsf{Conform}(a, x, t) = 1 \Leftrightarrow \mathsf{ObservedDep}(a) = \mathsf{Dep}^{\mathrm{inst}}(a). \tag{10}$$

This conformance predicate matters to ARETABA because no runtime authority object can be considered well-formed if its evaluation depends on hidden or undeclared inputs.

5.4 Authority Object in OTANIS-Compatible Form

OTANIS defines the runtime authority object as

$$\beta = (p, \sigma, b, \lambda, \rho, \kappa, \varepsilon, \pi_b, \tau_b, \omega) \tag{11}$$

where:

- p is the originating principal,
- σ is the authorised action-class scope,
- b is the boundary definition,
- λ defines lifecycle validity,
- ρ defines revocation,
- κ defines refusal and safe halt behaviour,
- ε defines escalation,
- π_b binds authority to provenance,
- τ_b defines traceability obligations,
- ω defines accountability binding.

The purpose of this paper is to specify how this object is constructed.

6 ARETABA as a Construction Discipline

6.1 Class-Level Synthesis and Runtime Resolution

The central construction idea is that OTANIS should not be forced to assume the existence of a valid runtime authority object. Instead, each action class should have a synthesised *authority template* that can be resolved deterministically at runtime.

Define the set of class-level authority templates as $\mathcal{B}_0$. Introduce the synthesis function

$$\mathsf{Synthesize} : \mathcal{A}_c \times R \to \mathcal{B}_0. \tag{12}$$

For action class α, define its class-level template:

$$\mathsf{AuthorityTemplate}(\alpha) := \beta^{\circ}_{\alpha} = \mathsf{Synthesize}(\alpha, R) \tag{13}$$

with

$$\beta^{\circ}_{\alpha} = (p_\alpha, \sigma_\alpha, b_\alpha, \lambda_\alpha, \rho_\alpha, \kappa_\alpha, \varepsilon_\alpha, \pi_{b,\alpha}, \tau_{b,\alpha}, \omega_\alpha). \tag{14}$$

The runtime authority object used by OTANIS is then the boundary-resolved instance:

$$\beta_e(a) := \mathsf{Resolve}(\beta^{\circ}_{\alpha(a)}, x_e(a), T^*_e(a)). \tag{15}$$

This distinction is important. β°_{α} is a class-level authority template. $\beta_e(a)$ is the concrete runtime authority object for action instance a, bound to the execution-boundary snapshot state $x_e(a)$ at $T^*_e(a)$.

ARETABA therefore answers two different questions:

- How is authority for a class α constructed and declared?
- How is that authority resolved for a concrete action instance a at runtime?

The synthesis function is therefore a governed construction step, not a semantic oracle. It may be implemented by humans, tools, or mixed workflows, but the framework does not claim

that Synthesize can infer missing domain truth, omitted precursor commitments, or undeclared authority semantics automatically. If synthesis is semantically wrong, the resulting authority template may remain internally neat while being wrong in reality. That failure lies upstream of runtime enforcement.

6.2 Authority and Scope

The authority component of ARETABA consists of the originating principal p_α and the authorised scope σ_α.

Define the principal binding as

$$p_\alpha := \mathsf{PrincipalBinding}(AS(\alpha)). \tag{16}$$

Define the class-level scope map as

$$\sigma_\alpha := \mathsf{ScopeMap}(S(\alpha), DS(\alpha), AS(\alpha)). \tag{17}$$

The purpose of σ_α is to make scope executable rather than descriptive. It is not enough to say that an action class is ‘‘in scope’’ in prose. The framework requires a deterministic scope object from which the runtime condition

$$\alpha(a) \in \sigma_\alpha \tag{18}$$

can be evaluated at the boundary.

This is structurally aligned with access-control work in which authority is treated as an explicit logical relation rather than an inferred intention [8--10]. What ARETABA adds is that the scope must remain bound to the governed irreversible action class at the execution boundary, not merely to an upstream identity or token.

6.3 Boundary Definition

The boundary component b_α is not simply a label naming a boundary. It is the full execution-boundary specification required to bind runtime authority to the correct commit surface.

Define

$$b_\alpha := \mathsf{BoundarySpec}(\mathsf{Cand}_\alpha(\alpha), Order(\alpha), \mathsf{SelectEarliest}_\alpha, IA(\alpha), EB(\alpha)). \tag{19}$$

This construction requirement reflects a central point of OTANIS. Runtime authority is only meaningful if it is bound to the correct governed irreversible boundary. If the boundary field is incomplete, ambiguous, later than the first governed irreversible effect, or non-bypass only on paper, then the authority object is not execution-valid even if its other fields are present.

In ARETABA terms, the boundary field must therefore be sufficient to support:

- deterministic binding to $T_e^*(a)$,
- evaluation of $\mathsf{Applicable}(b, x, t)$,
- commit uniqueness and mediation,
- refusal under boundary ambiguity,
- replayability of the selection decision.

Where that structure is absent, runtime governance is under-specified.

6.4 Lifecycle Validity

Lifecycle validity captures the fact that authority is not timeless. A runtime authority object may be structurally correct and still invalid because it has expired, because its contextual preconditions no longer hold, or because the evidence it depends on has drifted outside freshness bounds.

Define the class-level lifecycle specification as

$$\lambda_\alpha := \mathsf{LifecycleSpec}(RF(\alpha), IA(\alpha), \mathsf{Dep}(\alpha)). \tag{20}$$

At runtime, lifecycle validity is evaluated by

$$\mathsf{Valid}_\lambda(\beta_e(a), x, t) \in \{0, 1\}. \tag{21}$$

A compliant lifecycle specification must define:

- expiry conditions,
- contextual validity conditions,
- freshness requirements,
- re-evaluation triggers if authority-relevant state changes before $\mathsf{Commit}^*(a)$.

This is one reason ARETABA is not reducible to a static access token. Execution-time authority in a critical action path is not only about *who* may act. It is also about *whether the authority is still valid now.*

6.5 Revocation

Revocation is often treated informally in AI system design, as though the existence of an upstream approval or scope declaration were sufficient. It is not. Any authority that may be withdrawn, suspended, superseded, invalidated by fraud signals, or made unsafe by policy change must support revocation semantics at runtime.

Define the class-level revocation sources as

$$\rho_\alpha := \mathsf{RevocationSources}(AS(\alpha), RF(\alpha), \mathsf{Dep}(\alpha)). \tag{22}$$

At runtime, revocation is evaluated by

$$\mathsf{Revoked}_\rho(\beta_e(a), x, t) = 1 \Leftrightarrow \exists r \in \rho_\alpha \text{ such that } r(x, t) = 1. \tag{23}$$

A defensible revocation design requires two things.

First, the revocation sources must be declared. A system that ‘‘would probably consult fraud’’ or ‘‘normally uses a registry’’ is not sufficient. The source set must be explicit.

Second, the revocation evaluation must be dependency-complete and fresh. If the relevant state is stale, unavailable, or no longer within the declared freshness bound, the correct behaviour is refusal or escalation, not heuristic continuation. This is consistent with fail-closed safety and with the general systems principle that stale control state near an irreversible transition is a

structural hazard [11, 12].

6.6 Refusal and Safe Halt Behaviour

Refusal is not a side-effect of failed permission. In ARETABA it is a first-class field because the runtime governance question is not only whether the action may commit. It is also what the system must do when that answer is negative or indeterminate.

Define the refusal policy

$$\kappa_\alpha := \mathsf{RefusalPolicy}(\alpha). \tag{24}$$

Let $\mathcal{F}_{fail}$ be the set of boundary-relevant failure categories for action class α, such as:

- out-of-scope request,
- stale lifecycle state,
- active revocation,
- dependency-conformance failure,
- boundary ambiguity,
- missing evidence,
- traceability failure,
- accountability failure,
- timeout or indeterminacy.

Then κ_α maps those failures to a declared outcome, for example:

$$\kappa_\alpha : \mathcal{F}_{fail} \to \{\text{refuse}, \text{halt}, \text{fallback}\}. \tag{25}$$

This design choice matters because a system can be formally fail-closed and still operationally unsafe if it has no declared refusal mode for partially staged local state. Where reversible pre-commit state exists, the refusal path may need compensating containment semantics of the kind familiar from sagas and local rollback structures [13]. Where the effect is genuinely irreversible, refusal must happen before the commit boundary, not after.

6.7 Escalation

Escalation is distinct from refusal. Refusal is the correct outcome when authority is absent, invalid, stale, or unverifiable. Escalation is the correct outcome when higher authority is required, when declared thresholds are exceeded, or when the situation remains admissible only through a higher-priority decision surface.

Define the class-level escalation policy

$$\varepsilon_\alpha := \mathsf{EscalationPolicy}(\alpha). \tag{26}$$

At runtime, the escalation surface is engaged if

$$\mathsf{Escalate}(a) = 1 \Leftrightarrow \varepsilon_\alpha(x_e(a), T_e^*(a)) \neq \bot. \tag{27}$$

Define the escalation-safety predicate

$$\begin{aligned}\mathsf{EscalationSafe}(\alpha) = 1 \Leftrightarrow{}& \text{the escalation graph induced by } \varepsilon_\alpha \text{ is acyclic} \\ &\wedge \text{ every escalation path terminates in a unique accountable authority surface} \\ &\wedge \text{ no escalation path creates an implicit bypass of } \kappa_\alpha \\ &\wedge \text{ every terminating surface is compatible with } \omega_\alpha.\end{aligned} \tag{28}$$

A compliant escalation policy must therefore specify:

- which conditions require escalation,
- to whom escalation must terminate,
- whether autonomous execution is suspended pending that decision,
- what evidence must be preserved for the escalation recipient,
- and the terminating accountable authority surface for each admissible escalation path.

This is particularly important in critical settings. If the authority chain terminates in a human decision surface, that surface must itself be attributable, context-bound, and auditable. Otherwise ''escalation'' degenerates into an informal override channel rather than a governed authority path.

6.8 Traceability

Traceability in ARETABA is not ordinary application logging. It is the requirement that the authority decision at the boundary be bound into an integrity-protected trace structure sufficient for deterministic replay and forensic reconstruction.

Define the class-level traceability schema as

$$\tau_{b,\alpha} := \mathsf{TraceSchema}(\alpha, v_R(\alpha), \mathsf{Dep}(\alpha)). \tag{29}$$

At runtime, OTANIS requires that traceability be evaluated through the predicate

$$\mathsf{Traceable}(\tau_b, \tau_r, a, x, t) \in \{0, 1\}. \tag{30}$$

For ARETABA, the critical point is that $\tau_{b,\alpha}$ must be declared as part of the authority template. It must specify the required snapshot and boundary-binding evidence, including where applicable:

- registry version,
- runtime artefact version,
- selection inputs x_{sel},
- dependency observations,
- policy and provenance digests,
- governed semantic intent binding,
- payload-binding evidence,

- any bundle semantics required on asynchronous mediated paths.

The underlying idea is consistent with audit-log integrity work and replay-oriented forensic design [14, 15]. In a governed execution-bearing system, traceability is part of the authority surface itself because a non-replayable authority claim is not fully auditable.

6.9 Accountability

Accountability is the field that binds the authority decision to a legitimate, attributable source of responsibility.

Define the class-level accountability binding as

$$\omega_\alpha := \mathsf{AccountabilitySpec}(AS(\alpha), p_\alpha, \varepsilon_\alpha). \tag{31}$$

At runtime, the relevant predicate is

$$\mathsf{Accountable}(\omega, a, x, t) \in \{0, 1\}. \tag{32}$$

The architectural purpose of ω is to ensure that the system does not merely know who proposed an action, but who is accountable for allowing that action to commit under the governing authority structure. This may include:

- the accountable authority source,
- the terminating decision surface,
- the escalation owner where escalation exists,
- the attribution rule for autonomous and semi-autonomous execution.

Without accountability, runtime authority becomes difficult to distinguish from runtime convenience. ARETABA therefore treats accountability as a mandatory construction field, not an optional governance extra.

6.10 Boundary-Evaluated Admissibility

OTANIS defines the ARETABA admissibility predicate as

$$\begin{aligned} \mathsf{AdmissibleARETABA}(a, \beta, x, t) = 1 \Leftrightarrow\ & (\alpha(a) \in \sigma) \\ & \wedge\ \mathsf{Applicable}(b, x, t) = 1 \\ & \wedge\ \mathsf{Traceable}(\tau_b, \tau_r, a, x, t) = 1 \\ & \wedge\ \mathsf{Accountable}(\omega, a, x, t) = 1 \\ & \wedge\ \mathsf{Conform}(a, x, t) = 1 \\ & \wedge\ \mathsf{FreshState}(\mathsf{Dep}^{\mathrm{inst}}(a), t, \Delta_{\mathrm{safe}}) = 1. \end{aligned} \tag{33}$$

This paper does not alter that predicate. Instead, it specifies the construction discipline that makes its fields meaningful. In other words, Eq. (33) is the runtime test. ARETABA as a framework specifies how σ, b, τ_b, and ω are synthesised and bound so that the runtime test is not operating over ad hoc or incomplete structures.

7 Formal Construction Properties

7.1 Dependency Completeness

For each field $f \in \{b, \lambda, \rho, \kappa, \varepsilon, \tau_b, \omega\}$, let $\mathsf{Dep}_f(\alpha)$ denote the set of state keys that field functionally depends upon. Define the required ARETABA dependency set as

$$\mathsf{RequiredDep}(\alpha) := \bigcup_{f \in \{b,\lambda,\rho,\kappa,\varepsilon,\tau_b,\omega\}} \mathsf{Dep}_f(\alpha). \tag{34}$$

This definition is important because a runtime authority template that omits some dependency required by revocation, lifecycle validity, or boundary applicability is incomplete even if its field labels are present.

7.2 Completeness

Define

$$\begin{aligned} \mathsf{CompleteARETABA}(\alpha) = 1 \Leftrightarrow\ & \mathsf{AllFieldsDefined}(\beta_\alpha^\circ) = 1 \\ & \wedge\ \mathsf{RequiredDep}(\alpha) \subseteq \mathsf{Dep}(\alpha) \\ & \wedge\ \alpha \in \sigma_\alpha \\ & \wedge\ EB(\alpha) = 1. \end{aligned} \tag{35}$$

This is a structural completeness notion. It proves only that the class-level authority template contains the required fields, that every declared functional dependency of those fields is represented in the boundary registry dependency scope, that the class is actually included in its own authorised scope, and that the action class has a governable execution boundary declaration.

It does not by itself prove semantic richness, strong boundary correctness in reality, escalation safety, or runtime-resolvable executability. Those stronger properties are deferred respectively to $\mathsf{ConsistentARETABA}(\alpha)$, $\mathsf{EscalationSafe}(\alpha)$, and $\mathsf{ExecutableARETABA}(\alpha)$. Completeness here is therefore construction completeness relative to the declared framework, not semantic omniscience or deployment sufficiency.

7.3 Consistency

Define

$$\mathsf{ConsistentARETABA}(\alpha) = 1 \Leftrightarrow \mathsf{NoConflict}(\beta_\alpha^\circ, ISDAIRE(\alpha), R) = 1 \wedge \mathsf{EscalationSafe}(\alpha) = 1. \tag{36}$$

A consistent ARETABA template must satisfy at least the following:

- scope does not authorise effects excluded by $S(\alpha)$ or $DS(\alpha)$,
- the boundary field does not place commitment later than the declared irreversibility model in $IA(\alpha)$,
- lifecycle and revocation semantics do not conflict with the declared authority source $AS(\alpha)$,
- the escalation graph induced by ε_α is acyclic, terminates in a unique accountable authority surface, and is compatible with ω_α,
- escalation does not create an implicit bypass of refusal,
- traceability and accountability bindings are compatible with the boundary and versioning

model,

- the dependency scope required by the authority template is consistent with the registry declaration.

Consistency is therefore a relation between the authority template, the ex-ante admissibility model, the escalation-termination structure, and the registry state.

7.4 Executability

A complete and consistent authority template is still insufficient if it cannot be enforced on a real governed commit path. Define

$$\mathsf{ExecutableARETABA}(\alpha) = 1 \Leftrightarrow \mathsf{CompleteARETABA}(\alpha) = 1 \wedge \mathsf{ConsistentARETABA}(\alpha) = 1 \wedge \mathsf{WellBound}(\alpha) = 1. \tag{37}$$

Here $\mathsf{WellBound}(\alpha) = 1$ means, in substance, that:

- $\mathsf{Cand}_\alpha(\alpha) \neq \emptyset$,
- the ordering semantics and $\mathsf{SelectEarliest}_\alpha$ are deterministic under declared inputs,
- the commit primitive for realised instances of the action class is uniquely mediated and non-bypass,
- traceability and logging can be coupled to the governed commit path,
- runtime resolution can occur inside the bounded latency and freshness envelope for the action class.

This property matters because a beautifully specified authority template that cannot be executed under runtime constraints is not a valid authority discipline for an autonomous irreversible action class.

7.5 Interpretive Boundary

Three points should be stated clearly.

First, $\mathsf{CompleteARETABA}(\alpha) = 1$ does not imply that the action class is wise, fair, clinically sound, or legally sufficient.

Second, $\mathsf{ConsistentARETABA}(\alpha) = 1$ does not imply that the declared fields are semantically rich enough to cover every real-world hazard.

Third, $\mathsf{ExecutableARETABA}(\alpha) = 1$ does not imply that the wider system is overall safe. It implies only that the action class has a structurally executable runtime authority object on a governed commit path.

These are deliberate claim boundaries. Stronger claims would not be defensible.

8 Runtime Semantics Inside OTANIS

8.1 Runtime Resolvability

A class-level authority template must be resolvable at runtime from the execution-boundary snapshot state and the declared dependency scope. Define the runtime resolvability condition

$$\mathsf{RuntimeResolvable}(a) = 1 \tag{38}$$

iff the template $\beta^{\circ}_{\alpha(a)}$ can be instantiated as $\beta_e(a)$ using only:

- the execution-boundary snapshot $x_e(a)$,
- the declared realised dependency scope $\mathsf{Dep}^{\mathrm{inst}}(a)$,
- the bound execution boundary $T^*_e(a)$,
- the version-bound registry and policy artefacts.

If runtime resolution requires undeclared data, unbounded inference, unverifiable reconstruction, or ambiguous boundary interpretation, then the authority object is not runtime-resolvable.

8.2 Version Binding, Snapshot State, and Determinacy

As in OTANIS, the runtime artefact version must match the registry version:

$$\mathsf{VersionMatch}(a) = 1 \Leftrightarrow v_{\mathrm{runtime}}(a) = v_R(\alpha(a)). \tag{39}$$

The execution-boundary decision must be evaluated over a deterministic snapshot state $x_e(a)$, not over an unconstrained live state. The runtime authority object is then

$$\beta_e(a) = \mathsf{Resolve}(\beta^{\circ}_{\alpha(a)}, x_e(a), T^*_e(a)). \tag{40}$$

If the snapshot cannot be constructed, if version conformance fails, or if authority resolution is indeterminate, then

$$\mathsf{Determinate}(a, \beta_e(a), x_e(a), T^*_e(a)) = 0 \tag{41}$$

and the action must refuse or escalate. This follows the same fail-closed logic that OTANIS applies to governed execution [3] and is also consistent with deterministic state-machine reasoning in fault-tolerant systems [16].

8.3 Permit Semantics

The full execution permission predicate remains the OTANIS predicate:

$$\begin{aligned}\mathsf{Permit}(a, \beta_e(a), x_e(a)) \Leftrightarrow\ & \mathsf{AdmissibleISDAIRE}(\alpha(a)) \\ & \wedge\ \mathsf{AdmissibleARETABA}(a, \beta_e(a), x_e(a), T^*_e(a)) \\ & \wedge\ \mathsf{Valid}_{\lambda}(\beta_e(a), x_e(a), T^*_e(a)) \\ & \wedge\ \neg\mathsf{Revoked}_{\rho}(\beta_e(a), x_e(a), T^*_e(a)) \\ & \wedge\ \mathsf{VersionMatch}(a) \\ & \wedge\ \mathsf{Determinate}(a, \beta_e(a), x_e(a), T^*_e(a)).\end{aligned} \tag{42}$$

This equation is included here for a specific reason. It shows where ARETABA sits inside OTANIS. The ARETABA paper does not redefine the integrated boundary permit relation. It explains what must be true of $\beta_e(a)$, σ, b, τ_b, and ω for this relation to be meaningful and enforceable.

8.4 Re-Evaluation Under Authority-Relevant State Change

A runtime authority object is not stable merely because it was valid at some earlier point in the workflow. If authority-relevant state changes between evaluation and commit, the system must re-evaluate immediately before $\mathsf{Commit}^*(a)$.

This is already consistent with OTANIS and is particularly important for ARETABA because the authority template explicitly includes lifecycle, revocation, and boundary applicability semantics. In the presence of late state changes, stale revocation, or boundary drift, a previously resolved authority object may no longer be valid. The correct response is re-evaluation or refusal, not optimistic continuation. This is also aligned with the long-established problem of race conditions between check and use [17].

8.5 Trace Binding and Replayability

Because ARETABA treats traceability as a first-class field, runtime permission must be coupled to a trace commitment sufficient for later replay. In practical OTANIS terms, this means the boundary decision must be bound to the snapshot object $\mathsf{Snap}(a)$ and, where asynchronous execution is used, to the portable commit bundle $\mathsf{Bundle}(a)$ on the mediated path [3].

The traceability field $\tau_{b,\alpha}$ is therefore not satisfied by generic application logging. It is satisfied only if the required evidence for reconstructing the authority decision at $T_e^*(a)$ is present, integrity-protected, and attributable. This is one reason the ARETABA framework belongs at the execution boundary rather than in a separate observability layer.

9 Insurance Payout Worked Example

This section extends the insurance payout example so that the role of ARETABA becomes concrete.

9.1 Action Class Definition

Consider the execution-bearing action class

$$\alpha = \texttt{ClaimSettlement}.$$

A simplified ex-ante ISDAIRE profile might be:

- **Intent** $I(\alpha)$: settle a validated insurance claim.
- **Scope** $S(\alpha)$: direct monetary payment for approved claims within the insurer's delegated settlement domain.
- **Domain Separation** $DS(\alpha)$: excludes litigation, sanctions exceptions, fraud investigation hold, and policy dispute workflows.
- **Authority Source** $AS(\alpha)$: insurer claims-settlement authority under the declared operational policy.
- **Irreversibility Awareness** $IA(\alpha)$: yes, because rail submission may create an economically irreversible effect.
- **Risk Framing** $RF(\alpha)$: thresholded settlement risk, with higher authority required above declared amounts or under uncertainty.
- **Execution Boundary** $EB(\alpha)$: mediated payment-rail submission boundary on a non-bypass payout path.

Assume the action class is admissible under Eq. (8).

9.2 Candidate Boundaries and Boundary Binding

Suppose the declared candidate set is

$$\mathsf{Cand}_\alpha(\texttt{ClaimSettlement}) = \{t_1 = \texttt{PayoutGatewaySubmit},\ t_2 = \texttt{PaymentRailSubmission}\}. \tag{43}$$

If the architecture determines that t_2 is the earliest governed irreversible execution boundary under the declared ordering semantics, then

$$T_e^*(a) = \texttt{PaymentRailSubmission}. \tag{44}$$

This matters because the authority object must be valid *there*, not merely upstream. A claim handler approval or a model confidence score earlier in the workflow is not enough if revocation, fraud state, sanctions status, account validity, or escalation thresholds may still change before t_2.

9.3 Synthesised ARETABA Template

The class-level authority template becomes:

$$\beta^\circ_{\texttt{ClaimSettlement}} = (p_\alpha, \sigma_\alpha, b_\alpha, \lambda_\alpha, \rho_\alpha, \kappa_\alpha, \varepsilon_\alpha, \pi_{b,\alpha}, \tau_{b,\alpha}, \omega_\alpha). \tag{45}$$

A defensible instantiation is as follows.

Principal and scope.

$$p_\alpha = \mathsf{PrincipalBinding}(AS(\alpha)), \qquad \sigma_\alpha = \{\texttt{ClaimSettlement}\}. \tag{46}$$

Boundary field.

$$b_\alpha = \mathsf{BoundarySpec}(\mathsf{Cand}_\alpha(\alpha), Order(\alpha), \mathsf{SelectEarliest}_\alpha, IA(\alpha), EB(\alpha)) \tag{47}$$

with the applicable governed boundary $T_e^*(a) = \texttt{PaymentRailSubmission}$.

Lifecycle field. λ_α may require:

- approved-claim status,
- payee identity match,
- sanction-screen result within freshness window,
- payment amount within delegated authority threshold,
- approval not expired or superseded.

Revocation field. ρ_α may include:

- fraud-hold registry,
- sanctions or watchlist update channel,
- claims-authority suspension registry,

- payment-account revocation source.

Refusal field. κ_α may specify refusal on:

- stale sanctions data,
- active fraud flag,
- dependency-conformance failure,
- account-number mismatch,
- boundary ambiguity,
- trace or accountability failure.

Escalation field. ε_α may require escalation when:

- settlement amount exceeds a declared threshold such as £50,000,
- the claim falls into a manual-review exception category,
- evidence is fresh enough to avoid refusal but uncertain enough to require higher authority.

Traceability field. $\tau_{b,\alpha}$ must require binding of at least:

- claim identifier,
- payee identifier,
- amount and currency,
- registry version,
- dependency observations,
- rule and provenance digests,
- final governed intent and payload binding on the mediated submission path.

Accountability field. ω_α must identify the accountable authority source for automated settlement and the escalation owner for threshold or uncertainty cases.

9.4 Runtime Resolution

At runtime, for a specific settlement attempt a, the system constructs the execution-boundary snapshot $x_e(a)$ and resolves:

$$\beta_e(a) = \mathsf{Resolve}(\beta^{\circ}_{\texttt{ClaimSettlement}}, x_e(a), T^{*}_{e}(a)). \tag{48}$$

The settlement is permitted only if Eq. (42) holds.

This means, operationally, that all of the following must still be true at payment-rail submission:

- the action instance is within the authorised settlement scope,
- the current path really is the governed boundary path,

- traceability and accountability are present,
- no undeclared dependency influenced the decision,
- state freshness remains within the safe bound,
- lifecycle validity still holds,
- revocation has not fired,
- version conformance and determinacy still hold.

9.5 Edge Cases and Failure Interpretation

Three edge cases clarify the boundaries of the framework.

Case 1. Fraud flag arrives after upstream approval but before rail submission. Suppose the claim was approved and a proposal exists, but the fraud registry updates before $T_e^*(a)$. If fraud is a declared revocation source inside ρ_α, then $\mathsf{Revoked}_\rho(\beta_e(a), x_e(a), T_e^*(a)) = 1$, and the settlement must refuse. This is an ARETABA success case.

Case 2. Sanctions data is stale at the boundary. If the sanctions result exists but lies outside the declared freshness window, then $\mathsf{FreshState}(\mathsf{Dep}^{\mathrm{inst}}(a), T_e^*(a), \Delta_{\mathrm{safe}}) = 0$, and the system must refuse or escalate. Again, this is an ARETABA success case because stale authority state is not silently treated as current.

Case 3. Fraud dependency was never declared in the template. This is the important boundary case. If the fraud dependency was semantically necessary but omitted from $\mathsf{Dep}(\alpha)$, ρ_α, and the reviewed action-class artefacts, the runtime framework may still enforce the declared specification faithfully. The failure here is not a runtime ARETABA enforcement failure. It is a design-time semantic-adequacy failure. This distinction is critical if the framework is to remain honest and defensible.

Case 4. Governance-relevant payload changes after verification. If the final payment account or routing changes after bundle verification on the mediated path and that transformation was not already included in the declared adapter and payload-binding contract, then strict execution-boundary conformance cannot be claimed for the action class. This is a mediation-boundary failure, not a successful authority construction.

10 Falsifiability Conditions

The framework should be falsifiable. The following conditions are sufficient to show that an action class or deployment is non-compliant with ARETABA as specified here.

F0 (Template Absence). There exists an execution-bearing action class α for which no class-level authority template β_α° exists, yet autonomous execution is permitted.

F1 (Construction Incompleteness). There exists an action class α such that $\mathsf{CompleteARETABA}(\alpha) = 0$, yet autonomous execution remains admissible.

F2 (Boundary Inconsistency). There exists an action class α such that the boundary field b_α is inconsistent with $IA(\alpha)$, ambiguous under the declared ordering semantics, or not coupled to a non-bypass governed commit path, yet runtime authority is treated as valid.

F3 (Unresolvable Runtime Authority). There exists an executed action instance a for which $\beta_e(a)$ cannot be resolved deterministically from the authority template, the execution-boundary snapshot, and the declared dependency scope.

F4 (Trace or Accountability Defect). There exists an executed action instance a for which traceability or accountability was absent, malformed, or not recoverable from the boundary artefacts, yet the system claims strict execution-boundary authority.

F5 (Refusal or Escalation Undefined). There exists a boundary-relevant failure mode or threshold condition for which no declared refusal or escalation semantics exist, yet the system continues toward commit.

F6 (Semantic Overclaim). A deployment claims that $\mathsf{CompleteARETABA}(\alpha) = 1$ or $\mathsf{ExecutableARETABA}(\alpha) = 1$ establishes overall safety, domain adequacy, or legal sufficiency. That claim exceeds the framework.

These falsifiability conditions are useful because they keep the argument bounded. They also make the framework reviewable in procurement or audit contexts without pretending to settle every domain question at once.

11 Relationship to ISDAIRE, OTANIS, GAG, and MGAG

The formal relationship can now be stated directly.

ISDAIRE defines the structural ex-ante admissibility of the action class:

$$\mathsf{AdmissibleISDAIRE}(\alpha) = 1. \tag{49}$$

This is a necessary upstream condition for ARETABA, not a claim that the action class is already semantically sufficient, boundary-correct in reality, or otherwise acceptable for autonomous execution in the broader sense.

ARETABA synthesises the class-level authority template:

$$\beta^{\circ}_{\alpha} = \mathsf{Synthesize}(\alpha, R). \tag{50}$$

ARETABA then resolves the runtime authority object:

$$\beta_e(a) = \mathsf{Resolve}(\beta^{\circ}_{\alpha(a)}, x_e(a), T^{*}_{e}(a)). \tag{51}$$

OTANIS uses that boundary-resolved object in the permit equation:

$$\mathsf{Permit}(a, \beta_e(a), x_e(a)). \tag{52}$$

Where provenance and composition matter, GAG and MGAG then preserve legitimacy across single-layer and multi-layer systems [3].

Read as an execution pipeline:

$$ISDAIRE(\alpha) \implies \beta^{\circ}_{\alpha} \implies \beta_e(a) \implies \mathsf{Permit}(a, \beta_e(a), x_e(a)) \implies \mathsf{ComposedAuthority}(a). \tag{53}$$

This relation makes the architectural division of labour precise:

- ISDAIRE says whether the action class is structurally admissible in principle.

- ARETABA says how its runtime authority must be built and resolved.
- OTANIS says how that resolved authority is enforced at the boundary.
- GAG and MGAG say how legitimacy survives composition and layering.

Eq. (53) should therefore be read as a structural dependency chain, not as a proof that upstream semantic truth has been established. If $\mathsf{AdmissibleISDAIRE}(\alpha) = 1$ rests on semantically weak, incomplete, or boundary-misclassified artefacts, the downstream template and runtime authority object may still be constructed and enforced faithfully. In that case the failure remains a design-time semantic-adequacy or boundary-discovery failure rather than an ARETABA runtime-enforcement failure.

That is the correct placement of the ARETABA framework. It is not a parallel governance theory. It is the execution-time authority discipline inside the larger OTANIS architecture.

12 Discussion

The main value of isolating ARETABA is analytical clarity.

Without a dedicated execution-time authority framework, there is a tendency to collapse several different questions into one:

- ex-ante admissibility,
- runtime permission,
- provenance validity,
- semantic adequacy of the action-class model,
- audit sufficiency after the fact.

ARETABA separates the authority-construction question from those other layers. This improves review quality because it allows an engineer, auditor, or procuring organisation to ask a more precise question:

> Is the runtime authority object for this execution-bearing action class actually constructed, dependency-complete, boundary-valid, attributable, traceable, and resolvable at the governed boundary?

That question is narrower than ‘‘is the whole system safe’’ and much more operationally useful.

A second value is that it exposes where failures really sit. If an authority object is incomplete, that is a construction failure. If the boundary is wrong, that is a boundary-specification failure. If the runtime state is stale or drifted, that is a boundary-evaluation failure. If the semantically necessary hazard was never represented in the template, that is a design-time semantic-adequacy failure. This separation is valuable because it avoids both under-claiming and over-claiming.

A third value is that it makes procurement and external review more rigorous. A vendor may say that their system ‘‘checks approvals’’ or ‘‘uses policy’’. ARETABA forces the stronger question of whether runtime authority is an actual executable object with boundary semantics, revocation, refusal, escalation, traceability, and accountability. In high-consequence agentic systems, that is a much more meaningful distinction.

13 Conclusion

This paper formalises ARETABA as the execution-time authority framework that sits inside OTANIS and complements ISDAIRE. Its purpose is not to redefine the integrated OTANIS architecture, but to isolate the narrower discipline that makes runtime authority executable.

The core claim is simple. Ex-ante admissibility alone is not enough. A governed action class also needs a class-level authority template that is complete, consistent, boundary-valid, dependency-complete, traceable, accountable, and runtime-resolvable. Without that structure, a permit predicate may exist in code while governance remains structurally incomplete in fact.

ARETABA contributes that missing construction layer. It defines:

- how authority templates are synthesised from governed artefacts,
- how runtime authority objects are resolved at $T_e^*(a)$,
- what completeness, consistency, and executability conditions apply,
- how refusal, escalation, traceability, and accountability belong inside the authority object rather than outside it.

The framework remains intentionally bounded. It does not discover omitted hazards, repair semantically incomplete action-class specifications, or certify that the wider system is safe in every sense. What it does provide is a defensible answer to a narrower and critical question, namely what execution-time authority must look like if irreversible actions in agentic AI systems are to be governed rather than merely observed.

Read strictly, that is the contribution. It is also enough to matter.

References

[1] J. H. Saltzer and M. D. Schroeder, ‘‘The protection of information in computer systems,’’ *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278--1308, 1975.

[2] M. Otani, ‘‘Executable governance at the point of irreversibility,’’ AI Consultant Insights (AICI), Technical Report, 2026.

[3] ------, ‘‘Executable governance under multiple irreversibility boundaries: Deterministic binding to the earliest governed irreversible execution boundary in composed ai systems,’’ AI Consultant Insights (AICI), Technical Report, March 2026.

[4] National Institute of Standards and Technology, ‘‘Artificial intelligence risk management framework (AI RMF 1.0),’’ NIST, Tech. Rep. NIST AI 100-1, 2023. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf

[5] ------, ‘‘AI RMF profile for generative AI,’’ NIST, Tech. Rep. NIST AI 600-1, 2024. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

[6] OpenAI, ‘‘Practices for governing agentic AI systems,’’ 2023. [Online]. Available: https://cdn.openai.com/papers/practices-for-governing-agentic-ai-systems.pdf

[7] L. Lamport, ‘‘Time, clocks, and the ordering of events in a distributed system,’’ *Communications of the ACM*, vol. 21, no. 7, pp. 558--565, 1978.

[8] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin, ‘‘A calculus for access control in distributed systems,’’ *ACM Transactions on Programming Languages and Systems*, vol. 15, no. 4, pp. 706--734, 1993.

[9] E. Yuan and J. Tong, ‘‘Attribute based access control (ABAC) for web services,’’ in *Proceedings of the IEEE International Conference on Web Services (ICWS)*, 2005.

[10] V. C. Hu, D. Ferraiolo, D. R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, ‘‘Guide to attribute based access control (ABAC) definition and considerations,’’ NIST, Tech. Rep. NIST Special Publication 800-162, 2014. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-162.pdf

[11] N. G. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety.* MIT Press, 2011.

[12] E. A. Lee, ‘‘Cyber-physical systems: Design challenges,’’ in *Proceedings of the IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, 2008.

[13] H. Garcia-Molina and K. Salem, ‘‘Sagas,’’ in *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data (SIGMOD ’87)*, 1987, pp. 249--259.

[14] B. Schneier and J. Kelsey, ‘‘Secure audit logs to support computer forensics,’’ *ACM Transactions on Information and System Security*, vol. 2, no. 2, pp. 159--176, 1999.

[15] K. M. Chandy and L. Lamport, ‘‘Distributed snapshots: Determining global states of distributed systems,’’ *ACM Transactions on Computer Systems*, vol. 3, no. 1, pp. 63--75, 1985.

[16] F. B. Schneider, ‘‘Implementing fault-tolerant services using the state machine approach: A tutorial,’’ *ACM Computing Surveys*, vol. 22, no. 4, pp. 299--319, 1990.

[17] M. Bishop and M. Dilger, ‘‘Checking for race conditions in file accesses,’’ *Computing Systems*, vol. 9, no. 2, pp. 131--152, 1996.