ISDAIRE: A Formal Specification and Admissibility Framework for Executable Governance of Agentic AI Systems

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

22 March 2026

Abstract

As AI systems move from bounded recommendation into execution-bearing agentic architectures, the primary governance problem changes. The central difficulty is no longer only whether a model can generate a plausible output. It is whether an action class has been specified with enough precision, authority binding, and boundary discipline to make runtime enforcement meaningful at the point of irreversibility. Prior work on executable governance formalised execution-boundary control and deterministic binding to the earliest governed irreversible execution boundary, but it did not claim to solve the separate problem of how admissible action classes should be constructed ex ante [1, 2]. This paper addresses that gap.

The paper introduces ISDAIRE as a formal specification and admissibility framework for governed action classes in agentic systems. ISDAIRE is presented not as a runtime control surface and not as a universal safety architecture, but as the ex-ante specification layer that defines whether an action class has been specified strongly enough for executable governance. The framework is expressed through formal artefacts for intent, scope, domain separation, authority source, irreversibility awareness, risk framing, and execution-boundary definition, together with explicit condition sets, dependency declarations, version binding, and review obligations. A domain synthesis function is introduced to map domain semantics into governed specification artefacts. To preserve consistency with OTANIS, the paper retains the canonical class-level admissibility predicate $\mathsf{AdmissibleISDAIRE}(\alpha)$ and introduces a narrower auxiliary predicate, $\mathsf{StructuralAdmissible}(\alpha)$, for the weaker property that the governed specification object is well formed, internally consistent, and authority bound. A stronger deployment-side predicate, $\mathsf{AutonomousAdmissible}(\alpha)$, is then reserved for cases where boundary completeness also holds. This paper does not claim that any of these predicates alone establishes semantic adequacy, safety, legal sufficiency, or deployment acceptability.

The paper distinguishes syntactic completeness from semantic adequacy. It argues that runtime enforcement can only be as sound as the specification artefacts it is given, and that omitted dependencies, omitted precursor boundaries, stale authority assumptions, or semantically weak action classes remain design failures even when runtime enforcement is correct. A worked insurance payout example demonstrates how ISDAIRE constructs a governed action class for claim settlement, and how failures in boundary definition, dependency declaration, or authority binding propagate into execution risk. The paper positions ISDAIRE as the formal admissibility layer beneath OTANIS and MGAG, while also introducing a narrower auxiliary predicate for specification-object soundness and a stronger deployment-side qualification for autonomous execution. In that form, it provides a more defensible basis for specification, review, procurement qualification, and audit of execution-bearing agentic systems.

Keywords: Agentic AI, admissibility, architectural governance, authority, irreversibility, ex-ante specification, auditability.

1 Introduction

AI governance literature often concentrates on principles, risk controls, model behaviour, and oversight processes. Those concerns remain important, but they are not sufficient once a system can initiate or materially advance irreversible external effects such as financial settlement, access-control change, legal commitment, service dispatch, or physical actuation [3--5]. In such systems, the central failure is frequently structural rather than purely inferential. A model may reason plausibly and still trigger an action class that was never specified with enough clarity for authority to be enforced legitimately at execution time.

Prior work on executable governance formalised the idea that governance for execution-bearing action classes must bind to the point of irreversibility rather than remain only in policy documents or post-hoc review [1]. OTANIS then integrated ex-ante admissibility, runtime authority enforcement, deterministic boundary binding, compositional provenance, and multi-layer governance into a declared-boundary execution-governance architecture [2]. However, OTANIS was intentionally explicit about a boundary in its own claim surface. It enforces declared predicates. It does not synthesise missing governance semantics, discover omitted boundaries, or repair semantically weak action classes after deployment.

This paper addresses that missing layer. ISDAIRE is introduced as a formal specification and admissibility framework for governed action classes. Its purpose is to define what must exist before runtime governance can even make sense. It does not decide whether a model is aligned in the broad philosophical sense. It does not prove domain correctness. It does not replace regulatory, legal, clinical, or operational judgement. Instead, it defines the minimum formal structure that a governed action class must possess if later execution-boundary governance is to be meaningful, reviewable, and auditable.

In this paper, $\mathsf{StructuralAdmissible}(\alpha)$ denotes the narrower auxiliary predicate that the governed specification object for action class α is well formed, internally consistent, and actually governed. By contrast, $\mathsf{AdmissibleISDAIRE}(\alpha)$ retains the canonical OTANIS meaning as the class-level admissibility predicate over the seven ex-ante governance elements and the irreversibility-consistency condition. Neither predicate implies semantic adequacy, safety in every circumstance, legal sufficiency, or deployment acceptability without further domain judgement. That distinction is essential to the theorem surface of the paper.

The framework presented here draws on established foundations rather than claiming novelty for their individual ingredients. It is grounded in complete mediation [6], executable policy enforcement [7], formal specification discipline [8,9], access-control and delegation calculi [10,11], distributed ordering [12], distributed snapshot and replay discipline [13], and systems safety thinking [14]. The contribution claimed in this paper is not that these bodies of work did not exist. The contribution is their integration into a unified ex-ante admissibility architecture for execution-bearing action classes in agentic AI systems, aligned to OTANIS notation and to deterministic execution-boundary enforcement.

The practical value of that contribution is direct. Engineers need a design target for action classes that may later be allowed to commit irreversible outcomes. Reviewers need a way to distinguish between a system that is merely policy-described and a system whose execution-bearing actions have been specified with enough rigour to be governed. Procuring organisations need a way to identify structurally non-admissible autonomy before deployment. Auditors need a way

to separate design failures in specification from enforcement failures at runtime. ISDAIRE is proposed as a formal answer to that narrower question.

2 What ISDAIRE Contributes as a Framework

ISDAIRE should be read as a formal specification framework for governed action classes, not as a runtime control plane. On its own, it does not establish runtime governability, physical deployability, or autonomous execution fitness. It defines the ex-ante artefacts that must exist before runtime governance can be more than an empty gate. The canonical class-level admissibility predicate remains $\mathsf{AdmissibleISDAIRE}(\alpha)$, consistent with OTANIS. This paper also introduces two additional distinctions that are useful analytically but must not be confused with that canonical meaning, namely $\mathsf{StructuralAdmissible}(\alpha)$ for the weaker specification-object property and $\mathsf{AutonomousAdmissible}(\alpha)$ for the stronger deployment-side qualification that additionally requires boundary completeness. In practical terms, ISDAIRE answers four questions.

First, what exactly is the action class the system is being allowed to attempt. Second, under which authority and within which effect boundaries can that action class operate. Third, where is the earliest governed irreversible boundary for that class, and what dependencies and conditions must be bound to it. Fourth, how can a reviewer distinguish between an action class that is merely described, one that is structurally sound as a governed specification object, and one that satisfies the canonical ex-ante admissibility contract for executable governance.

This contribution is narrower than a claim to comprehensive system safety. It is also stronger in another sense, because it gives a precise object for design review. Rather than asking whether a whole system appears well governed in general, ISDAIRE asks whether each execution-bearing action class has a governed specification that is well formed, consistent, authority bound, versioned, and compatible with runtime execution-boundary control.

In this paper, ISDAIRE is used to refer to seven ex-ante specification obligations over an action class α:

1. intent $I(\alpha)$,
2. scope $S(\alpha)$,
3. domain separation $DS(\alpha)$,
4. authority source $AS(\alpha)$,
5. irreversibility awareness $IA(\alpha)$,
6. risk framing $RF(\alpha)$,
7. execution-boundary definition $EB(\alpha)$.

These seven obligations are not sufficient by themselves to make execution safe. They become operational only when paired with explicit condition sets, dependency declarations, version binding, and review processes. For that reason the formal synthesis function in this paper maps an action class into a larger governed specification object, not merely into seven booleans.

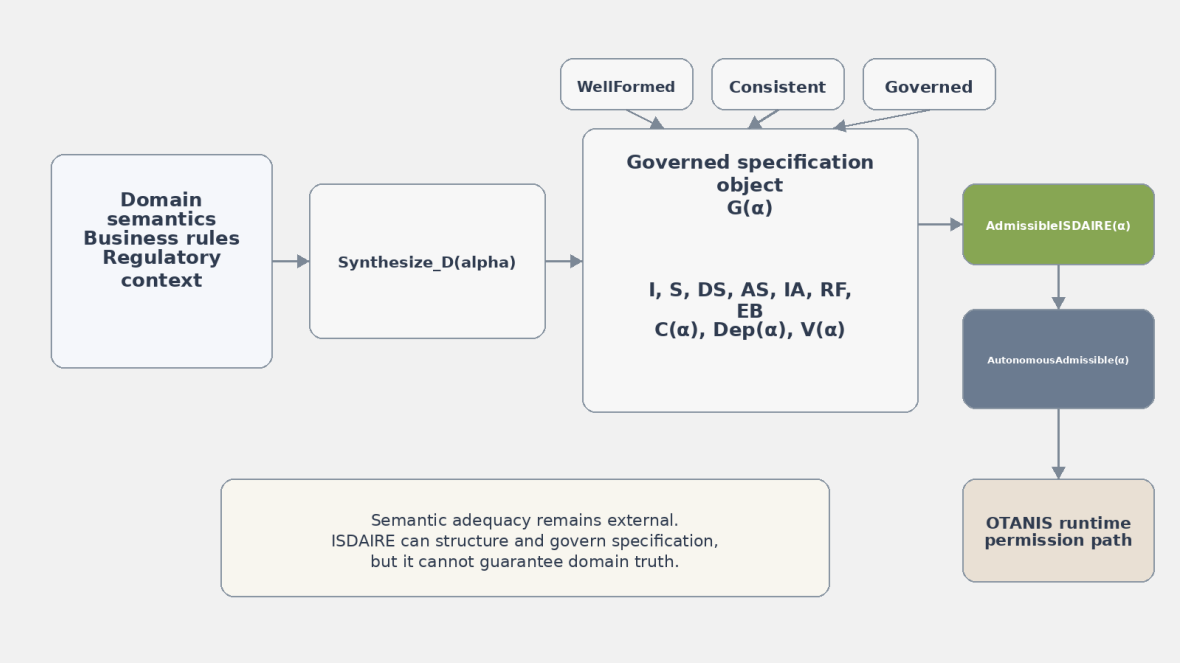

Figure 1: ISDAIRE in the governance stack. Domain semantics are synthesised into a governed specification object $G(\alpha)$. The paper introduces $\mathsf{StructuralAdmissible}(\alpha)$ as an auxiliary predicate for specification-object soundness, retains the canonical OTANIS class-level predicate $\mathsf{AdmissibleISDAIRE}(\alpha)$, and uses the stronger deployment-side qualification $\mathsf{AutonomousAdmissible}(\alpha)$ when boundary completeness is additionally required. Semantic adequacy remains external to the formal specification layer.

3 Scope and Non-Claims

3.1 Scope

This paper applies to execution-bearing action classes in agentic systems when the class can lead directly or compositionally to an irreversible external effect and where the architecture aspires to runtime governance under a declared candidate-boundary model with a runtime-bound governed execution boundary. The unit of analysis is therefore the governed action class and its realised commit path, not the whole system in the abstract.

The framework is relevant to sectors such as finance, insurance, healthcare, logistics, cyber operations, and industrial control, but it is not sector-specific. Applicability depends on the structure of the action class, not on the label attached to the industry.

3.2 Out of Scope

Purely advisory outputs, offline analytics, batch scoring that does not itself commit external effects, and unconstrained action proposals that are never intended to be executed autonomously are out of scope. Such systems may still be important, but they do not raise the same execution-boundary admissibility question.

3.3 Non-Claims

ISDAIRE does not guarantee correctness, safety, fairness, legal sufficiency, domain validity, or regulatory compliance. It does not claim to discover omitted hazards automatically. It does not infer the true irreversible boundary from arbitrary infrastructure at runtime. It does not prove that a condition set is semantically adequate for all real-world harms. It does not turn

a probabilistic upstream agent into a trustworthy governor merely by wrapping it in policy language [3, 7].

Its claim is narrower and more defensible. ISDAIRE specifies what a governed action class must contain before OTANIS-style runtime governance can even be meaningfully asserted. If those ex-ante artefacts are missing, contradictory, stale, or semantically weak, runtime enforcement may still be implemented correctly and yet the deployment remains poorly governed. That is a design-time or specification failure, not an execution-boundary enforcement success.

4 Architectural Premise

The framework in this paper is built on the following premise.

P1. Runtime governance of an execution-bearing action class is meaningful only if the action class has first been specified as a governed object whose authority source, effect scope, boundary semantics, and dependency conditions can be stated, checked, reviewed, and version-bound before deployment.

This premise follows directly from complete mediation and from the theory of enforceable policies. A runtime gate can only mediate properties that have been expressed in a form the gate can evaluate [6, 7]. If an action class has no explicit authority source, no clear effect scope, no declared boundary, or no declared dependency set, then the runtime layer is not governing the action in any meaningful sense. It is merely checking whatever fragment of semantics happened to be encoded.

The consequence is practical. Governance for agentic execution does not begin only when a realised run has already been bound to $T_e^*(a)$. It begins when an organisation decides whether a class of actions is admissible for governed execution at all and how candidate execution-boundary events $T_e(a)$ are to be declared, ordered, and governed.

5 Problem Statement

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

In OTANIS, execution-time permission for an action instance depends, among other predicates, on the class-level admissibility condition $\mathsf{AdmissibleISDAIRE}(\alpha(a))$ [2]. Informally, that means runtime governance assumes the class has already been specified well enough to be governed.

The problem is that a conjunction of element names is not itself a construction method. If one writes

$$I(\alpha) \wedge S(\alpha) \wedge DS(\alpha) \wedge AS(\alpha) \wedge IA(\alpha) \wedge RF(\alpha) \wedge EB(\alpha), \tag{1}$$

one has named the pieces of admissibility, but one has not yet said how those pieces are derived, typed, reviewed, versioned, or checked for completeness and contradiction.

That gap matters because runtime control cannot compensate for weak ex-ante semantics. If the declared action class omits a fraud dependency, misidentifies the earliest irreversible boundary, allows an authority object that is too broad, or treats a legally binding precursor as merely preparatory, then the runtime layer may enforce the wrong design perfectly.

The central question is therefore this.

Q1. What formal structure must exist for an action class α before it can be considered admissible for executable governance, and how should that structure be distinguished from

weaker specification-object soundness and stronger autonomous-deployment qualification?

This paper answers that question by introducing a domain synthesis function from an action class into a governed specification object, by preserving the canonical OTANIS definition of $\mathsf{AdmissibleISDAIRE}(\alpha)$, and by defining separate auxiliary predicates for specification-object soundness and stronger autonomous qualification.

6 Formal Model and Notation

6.1 System Model

Let an agentic deployment be represented by a directed graph

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

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

Let T denote a set of execution events equipped with a partial order $\prec$ representing causal precedence [12]. Let $\mathcal{P}_{fin}(T)$ denote the family of all finite subsets of T.

Define a generic execution-boundary function

$$T_e : \mathcal{I} \to T \tag{3}$$

where $T_e(a)$ denotes a generic candidate execution-substrate commit event at which action instance a could become irreversible.

Let X denote the space of system states relevant to governance evaluation, and let $\mathsf{Keys}(x)$ denote the addressable state keys of a realised state $x \in X$.

6.2 Candidate Boundary Set and Runtime Boundary Binding

The generic function $T_e(a)$ names a candidate execution-substrate commit event for action instance a. In composed or route-varying systems, an action instance may expose multiple such candidates. ISDAIRE therefore works not with a single assumed boundary, but with an ex-ante declared candidate set from which the realised governed boundary is bound at runtime.

Define a declared finite candidate set function over action classes

$$\mathsf{Cand}_\alpha : \mathcal{A}_c \to \mathcal{P}_{fin}(T) \tag{4}$$

where $\mathsf{Cand}_\alpha(\alpha)$ is the ex-ante declared finite non-empty set of candidate execution events for action class α.

For a runtime action instance $a \in \mathcal{I}$, define the instance candidate set as

$$\mathsf{Cand}(a) := \mathsf{Cand}_\alpha(\alpha(a)). \tag{5}$$

Define a deterministic selection function

$$\mathsf{SelectEarliest} : \mathcal{P}_{fin}(T) \times X \to T \tag{6}$$

where X denotes the declared runtime selection inputs used for boundary binding.

For action-class-specific registry references, $\mathsf{SelectEarliest}_\alpha$ denotes the declared selection contract associated with action class α.

We distinguish three objects that must not be conflated. First, $x_{sel} \in X$ denotes the realised selection inputs used only for boundary binding via $\mathsf{SelectEarliest}(\mathsf{Cand}(a), x_{sel})$. Second, x denotes system state used to evaluate admissibility and later runtime governance predicates. Third, $x_e(a)$ denotes the execution-boundary snapshot state actually used at the bound boundary.

For each action class α, the boundary model must also declare an ordering function

$$Order(\alpha) : \mathsf{Cand}_\alpha(\alpha) \times \mathsf{Cand}_\alpha(\alpha) \times X \to \{-1, 0, 1\} \tag{7}$$

that induces, for each realised run under declared inputs x_{sel}, a total preorder together with a deterministic tie-break rule.

The generic function $T_e(a)$ denotes a candidate execution-boundary event. The bound earliest governed execution boundary is then

$$T_e^*(a) := \mathsf{SelectEarliest}(\mathsf{Cand}(a), x_{sel}(a)). \tag{8}$$

Accordingly, $T_e(a)$ is generic and candidate-level, while $T_e^*(a)$ is the unique runtime-bound earliest governed execution boundary for the realised run under declared ordering semantics and declared selection inputs.

In this paper, ‘‘earliest’’ means earliest under the declared ordering semantics for the action class and the realised declared inputs. It does not mean absolute wall-clock priority across arbitrary infrastructures. If the ordering model does not induce a deterministic result for the realised run, autonomous execution is non-admissible for that action instance under default-deny semantics.

6.3 Governed Specification Space

For each action class $\alpha \in \mathcal{A}_c$, define a governed specification object

$$G(\alpha) = \big(I(\alpha), S(\alpha), DS(\alpha), AS(\alpha), IA(\alpha), RF(\alpha), EB(\alpha), C(\alpha), \mathsf{Dep}(\alpha), V(\alpha)\big) \tag{9}$$

where:

- $I(\alpha)$ is the intent artefact,
- $S(\alpha)$ is the scope artefact,
- $DS(\alpha)$ is the domain-separation artefact,
- $AS(\alpha)$ is the authority-source artefact,
- $IA(\alpha)$ is the irreversibility-awareness artefact,
- $RF(\alpha)$ is the risk-framing artefact,
- $EB(\alpha)$ is the execution-boundary artefact,
- $C(\alpha)$ is the finite condition set,
- $\mathsf{Dep}(\alpha)$ is the declared dependency scope,
- $V(\alpha)$ is the governance version and integrity-binding metadata.

The inclusion of $C(\alpha)$, $\mathsf{Dep}(\alpha)$, and $V(\alpha)$ is deliberate. Without them, the seven named elements remain too abstract to support deterministic runtime enforcement.

6.4 Domain Synthesis Function

Let D denote a domain with its own business semantics, regulatory constraints, effect vocabulary, operational boundaries, and authority model. Define a domain synthesis function

$$\mathsf{Synthesize}_D : \mathcal{A}_c \to \mathcal{G} \tag{10}$$

where $\mathcal{G}$ is the space of governed specification objects and

$$\mathsf{Synthesize}_D(\alpha) = G(\alpha). \tag{11}$$

$\mathsf{Synthesize}_D$ is not a claim that admissibility can be generated automatically from prose. It denotes the governed process, human, machine-assisted, or mixed, by which domain semantics are transformed into machine-legible and reviewable action-class artefacts.

6.5 Well-Formedness, Consistency, and Governance Binding

Define the following predicates over $G(\alpha)$.

Well-formedness

$$\mathsf{WellFormed}(G(\alpha)) = 1 \tag{12}$$

iff every required artefact exists, is typed, passes schema validation, and is sufficiently structured for the intended runtime gate.

Consistency

$$\mathsf{Consistent}(G(\alpha)) = 1 \tag{13}$$

iff the artefacts are not mutually contradictory. Examples of inconsistency include a scope that authorises a transition into a region forbidden by domain separation, an authority source whose scope does not cover the declared action class, or an execution boundary that lies after a declared irreversible precursor.

Governance binding

$$\mathsf{Governed}(G(\alpha)) = 1 \tag{14}$$

iff the artefacts are authority bound, versioned, integrity protected, reviewable, and subject to explicit change control.

6.6 Structural Specification Soundness

To preserve notation consistency with OTANIS, this paper introduces a separate auxiliary predicate for the narrower specification-object property developed here:

$$\mathsf{StructuralAdmissible}(\alpha) = 1 \Leftrightarrow \mathsf{WellFormed}(G(\alpha)) \wedge \mathsf{Consistent}(G(\alpha)) \wedge \mathsf{Governed}(G(\alpha)). \tag{15}$$

$\mathsf{StructuralAdmissible}(\alpha) = 1$ means only that the governed specification object for action class α exists in a structurally usable form, is internally coherent, and is actually under authority-bound governance. It does not by itself imply canonical ISDAIRE admissibility, semantic adequacy, safety, legal sufficiency, or deployment acceptability. In particular,

$$\mathsf{StructuralAdmissible}(\alpha) = 1 \not\Rightarrow \mathsf{Adequate}(\alpha, D).$$

This predicate is introduced only to name the weaker property that the current paper was trying to isolate. It is not a replacement for $\mathsf{AdmissibleISDAIRE}(\alpha)$.

6.7 Canonical ISDAIRE Admissibility Definition

To remain fully consistent with OTANIS and the multi-boundary execution paper, this paper retains the canonical ex-ante admissibility structure:

$$\begin{aligned} ISDAIRE(\alpha) := I(\alpha) \wedge S(\alpha) \wedge DS(\alpha) \wedge AS(\alpha) \\ \wedge IA(\alpha) \wedge RF(\alpha) \wedge EB(\alpha). \end{aligned} \tag{16}$$

Define $\mathsf{AdmissibleISDAIRE}(\alpha) \in \{0, 1\}$ as

$$\begin{aligned} \mathsf{AdmissibleISDAIRE}(\alpha) = 1 \Leftrightarrow\ & I(\alpha) = 1 \\ & \wedge\ S(\alpha) = 1 \\ & \wedge\ DS(\alpha) = 1 \\ & \wedge\ AS(\alpha) = 1 \\ & \wedge\ IA(\alpha) = 1 \\ & \wedge\ RF(\alpha) = 1 \\ & \wedge\ EB(\alpha) = 1 \\ & \wedge\ \mathsf{Consistent}(EB(\alpha), IA(\alpha)) = 1. \end{aligned} \tag{17}$$

Here $P(\alpha) = 1$ for $P \in \{I, S, DS, AS, IA, RF, EB\}$ means that the corresponding governed specification artefact for action class α exists, is valid under the declared authority source, is machine-legible where required by the runtime gate, and passes any integrity and schema checks required by the deployment. This is therefore a formal admissibility predicate over governed specification artefacts, not an informal human impression encoded as algebra.

The purpose of $\mathsf{AdmissibleISDAIRE}(\alpha)$ is not to pretend that governance judgement has been mathematically solved. Its purpose is to provide the same fail-closed class-level indicator used by OTANIS, namely whether the required ex-ante governed specification state exists and is internally coherent for action class α. As in OTANIS, this does not imply semantic adequacy, safety, legal sufficiency, or deployment acceptability.

Interpretive note. In the multi-boundary model, ex-ante admissibility requires the action class to have declared candidate boundaries, deterministic ordering semantics, declared dependency scope, explicit authority source, and declared irreversibility classification in the approved boundary registry. Runtime permission is a separate question and is not implied by ex-ante admissibility alone. If a materially earlier irreversible or binding precursor event has been omitted from $\mathsf{Cand}_\alpha(\alpha)$, the action class may still appear well specified at a superficial level while remaining semantically invalid for governed execution.

6.8 Autonomous Execution Qualification

For autonomous execution, this paper introduces a stronger deployment-side qualification than canonical class-level admissibility alone:

$$\mathsf{AutonomousAdmissible}(\alpha) = 1 \Leftrightarrow \mathsf{AdmissibleISDAIRE}(\alpha) \wedge \mathsf{BoundaryComplete}(\alpha). \tag{18}$$

Here $\mathsf{BoundaryComplete}(\alpha)$ is the stronger boundary-completeness property defined later in Section 10.3. The purpose of Eq. (18) is to separate two questions that should not be collapsed:

1. whether the class satisfies the canonical OTANIS admissibility predicate at all, and

2. whether that admissible class is also strong enough, at the precursor and boundary level, to support autonomous execution under the stricter qualification used in this paper.

Accordingly,

$$\mathsf{AutonomousAdmissible}(\alpha) = 1 \Rightarrow \mathsf{AdmissibleISDAIRE}(\alpha) = 1,$$

but not conversely. A class may satisfy $\mathsf{AdmissibleISDAIRE}(\alpha) = 1$ and still fail $\mathsf{AutonomousAdmissible}(\alpha) = 1$ because precursor semantics remain unresolved or because boundary-completeness evidence remains insufficient.

6.9 Boundary Compatibility with OTANIS

Because the present paper is aligned to OTANIS notation, $EB(\alpha)$ must be treated as a deployment-level enforceability predicate rather than as a merely descriptive label. Accordingly,

$$EB(\alpha) = 1 \Leftrightarrow \begin{cases} \mathsf{Cand}_\alpha(\alpha) \neq \emptyset, \\ \mathsf{Cand}_\alpha(\alpha) \text{ is finite}, \\ \mathsf{SelectEarliest}_\alpha \text{ is deterministic under declared inputs}, \\ \text{the runtime implementation is equivalent to the declared selection contract}, \\ \text{the bound boundary is interceptable and mediated}, \\ \text{atomic enforcement is available at the bound boundary}. \end{cases} \tag{19}$$

Equivalently, $EB(\alpha) = 1$ only if all of the following hold for action class α:

1. a finite non-empty candidate boundary set $\mathsf{Cand}_\alpha(\alpha) \subseteq T$ is declared,
2. a deterministic selection contract $\mathsf{SelectEarliest}_\alpha$ is declared,
3. the declared ordering semantics do not contradict causal precedence $\prec$,
4. the selected bound boundary can be intercepted and mediated,
5. no declared irreversible effect encoded by $IA(\alpha)$ is reachable on a bypass path that escapes equivalent governance.

$EB(\alpha)$ depends not only on the semantic classification of irreversibility but also on the technical properties of the execution substrate available for that action class. Therefore $EB(\alpha)$ may differ across deployments of the same abstract action class. If no economically or physically irreversible commit boundary exists for α, or if the earliest such boundary cannot be intercepted and atomically coupled to governance evaluation, then $EB(\alpha) = 0$ and the model is not applicable to that action class for autonomous execution.

Accordingly, if these conditions fail, then $EB(\alpha) = 0$ and therefore $\mathsf{AutonomousAdmissible}(\alpha) = 0$ for OTANIS-compatible autonomous execution [2].

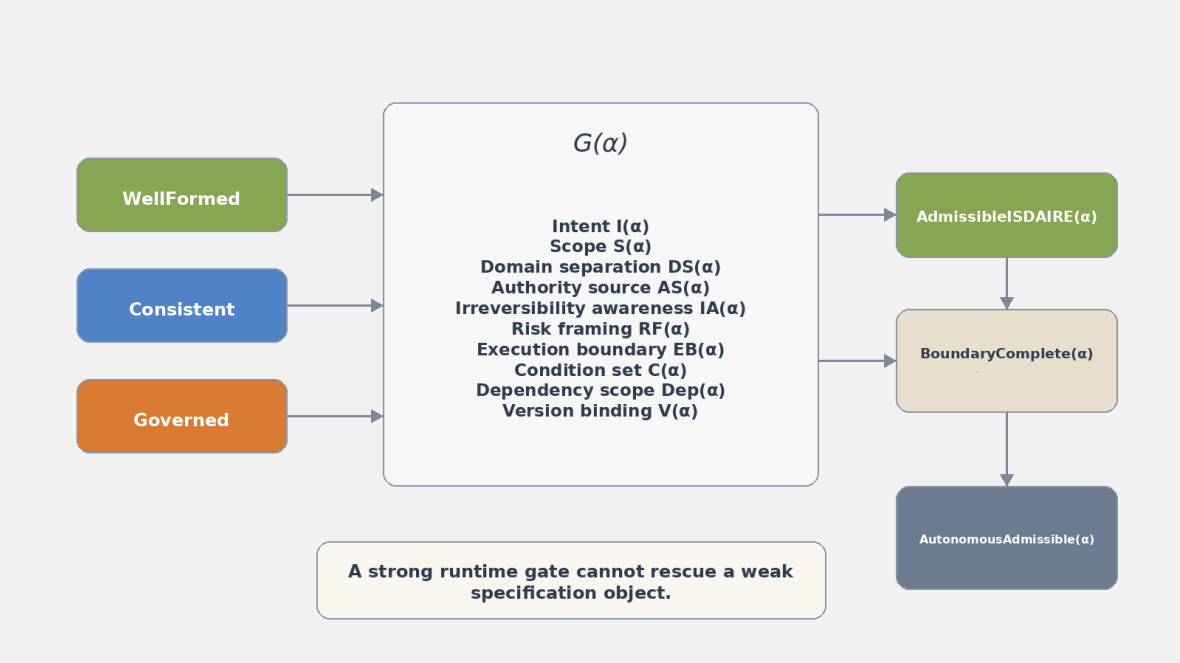

Figure 2: Governed specification object $G(\alpha)$ and the admissibility hierarchy. Structural admissibility requires the specification object to be well formed, internally consistent, and actually governed. Autonomous execution readiness is stricter, requiring $\mathsf{BoundaryComplete}(\alpha)$ in addition to canonical $\mathsf{AdmissibleISDAIRE}(\alpha)$.

7 The Seven ISDAIRE Elements

This section defines each element as a governed specification obligation rather than as an informal concept.

7.1 Intent $I(\alpha)$

Intent specifies what the action class is for. It is not the raw natural-language request from a user and not the model's private intermediate reasoning. It is the governed purpose declaration against which later execution is judged.

Let $\mathcal{J}$ denote the domain intent space. Then

$$I(\alpha) \subseteq \mathcal{J}. \tag{20}$$

For an action instance a, the realised intent label must satisfy

$$\mathsf{Intent}(a) \in I(\alpha(a)). \tag{21}$$

A compliant intent artefact must at minimum:

1. define the permitted purpose class,
2. distinguish the class from nearby but materially different intents,
3. support machine-legible matching or classification,
4. be authorised by an appropriate source under $AS(\alpha)$.

Intent is often underspecified in weak deployments. A class named *ProcessClaim* may conceal materially different actions such as acknowledge receipt, approve claim, schedule manual review, or submit settlement. If those are not separated, later governance becomes vague before runtime even begins.

7.2 Scope $S(\alpha)$

Scope defines the admissible state-transition envelope for the action class. It answers what kind of effect transition the action class may legitimately induce.

Let X_{sys} be the relevant state space for the action class. Then

$$S(\alpha) \subseteq X_{sys} \times X_{sys}. \tag{22}$$

For an action instance a with pre-state x_{pre} and post-state x_{post},

$$(x_{pre}, x_{post}) \in S(\alpha(a)) \tag{23}$$

must hold for the effect to remain in scope.

Scope is narrower than business desirability and broader than a single field update. It should capture what kind of state transition is permitted in principle. In an insurance domain, *approved-claim* $\rightarrow$ *paid-claim* may be in scope for one action class, while *fraud-flagged-claim* $\rightarrow$ *paid-claim* is not.

7.3 Domain Separation $DS(\alpha)$

Domain separation identifies effect regions, entities, situations, or state zones that the action class must not cross. It prevents overbroad action classes from silently expanding into forbidden domains.

Let R_{forbid} denote the set of forbidden regions in the relevant domain state space. Then

$$DS(\alpha) \subseteq R_{forbid}. \tag{24}$$

For a realised action instance a, if the execution context enters a forbidden region, the class is non-admissible:

$$x_e(a) \in DS(\alpha(a)) \Rightarrow \mathsf{AdmissibleISDAIRE}(\alpha(a)) = 0. \tag{25}$$

Domain separation may refer to entities under legal hold, sanctioned counterparties, fraud-investigation queues, excluded jurisdictions, safety-critical actuator modes, or any other region the class must not enter. It is not redundant with scope. Scope says what class of transition is allowed. Domain separation says where that class must never operate.

7.4 Authority Source $AS(\alpha)$

Authority source identifies who or what may authorise the action class, and under which delegation semantics. It does not merely list roles. It binds action admissibility to accountable principals, issuers, or approval structures.

Let $\mathcal{P}$ denote the set of principals and recognised authorising entities. Then

$$AS(\alpha) \subseteq \mathcal{P}. \tag{26}$$

For a resolved authority object β later used at runtime, admissibility requires its origin to lie

within the declared source class:

$$\mathsf{Authority}(\beta) \in AS(\alpha). \tag{27}$$

Where delegation is permitted, the delegation chain must be defined in a way consistent with formal access-control and trust-calculus reasoning [10, 11]. An action class with no explicit authority source is structurally non-admissible, regardless of whether the implementation later checks a token.

7.5 Irreversibility Awareness $IA(\alpha)$

Irreversibility awareness defines whether the action class can produce, directly or compositionally, an external effect that cannot simply be undone by local rollback. It is broader than a binary label, although a binary indicator remains useful.

Define a classification function

$$r(\alpha) : \mathcal{A}_c \to \{0, 1\} \tag{28}$$

where $r(\alpha) = 1$ denotes an execution-bearing class with irreversible external consequences.

For richer specification, define

$$IA(\alpha) = (r(\alpha), IE_{decl}(\alpha), \mathsf{Cand}_\alpha(\alpha), Order(\alpha)) \tag{29}$$

where:

- $IE_{decl}(\alpha)$ is the declared set of irreversible external effects,
- $\mathsf{Cand}_\alpha(\alpha)$ is the declared candidate boundary set,
- $Order(\alpha)$ is the declared ordering model used in boundary selection.

This definition matters because a binary *yes or no* marker alone is not enough to govern a real action class. What matters architecturally is which irreversible effects are being claimed, which precursor transitions may already bind the organisation, and whether the earliest governed boundary truly dominates those effects [2, 14].

7.6 Risk Framing $RF(\alpha)$

Risk framing defines how the structural admissibility of the action class is conditioned by impact, uncertainty, thresholds, escalation rules, refusal triggers, and consequence assumptions. It is not only a score. It is the governed semantics of risk for the class.

A useful formalisation is:

$$RF(\alpha) = (\mu_\alpha, \tau_\alpha, \varepsilon_\alpha, \kappa_\alpha, \phi_\alpha) \tag{30}$$

where:

- μ_α is the declared risk measure or set of measures,
- τ_α is the threshold structure,
- ε_α defines escalation conditions,
- κ_α defines refusal or halt conditions arising from risk-state failure,
- ϕ_α records the declared consequence classes, freshness assumptions, and external override

conditions that bound interpretation of the risk frame.

For a realised execution state x, one may write

$$\mathsf{Risk}_\alpha(x) \leq \tau_\alpha \tag{31}$$

as a simple admissibility condition, but in many domains the risk frame is vector-valued or rule-based rather than scalar.

For a high-consequence execution-bearing action class, a compliant $RF(\alpha)$ artefact must declare at minimum:

1. which consequence classes the frame is intended to control,
2. which variables, evidence sources, and freshness assumptions feed the frame,
3. which thresholds trigger continuation, escalation, refusal, or manual review,
4. which legal, compliance, or domain constraints override the quantitative frame,
5. whether correlated loss, regime change, or coupled downstream effects are explicitly modelled or explicitly excluded.

A class whose risk frame consists only of a score and threshold may be typed and well formed in a weak sense while remaining structurally poor in a governance sense. What matters is that the risk frame is declared ex ante, bounded, reviewable, and severe enough to support later execution-boundary governance rather than being smuggled into runtime as an opaque heuristic [3, 4].

7.7 Execution Boundary $EB(\alpha)$

Execution-boundary definition specifies where the class becomes governed for purposes of runtime enforcement. This paper reuses OTANIS notation directly and applies it at the action-class specification layer. For notation parity with the multi-boundary model, $T_e(a)$ denotes a generic candidate execution-boundary event for action instance a, while $T_e^*(a)$ denotes the unique bound earliest governed execution boundary selected at runtime from the declared candidate set $\mathsf{Cand}(a)$.

Let

$$\mathsf{Cand}_\alpha : \mathcal{A}_c \to \mathcal{P}_{fin}(T) \tag{32}$$

map an action class to its finite non-empty candidate boundary set. For a realised action instance a,

$$\mathsf{Cand}(a) := \mathsf{Cand}_\alpha(\alpha(a)). \tag{33}$$

Define a deterministic selection function

$$\mathsf{SelectEarliest} : \mathcal{P}_{fin}(T) \times X \to T \tag{34}$$

and the bound earliest governed execution boundary

$$T_e^*(a) := \mathsf{SelectEarliest}(\mathsf{Cand}(a), x_{sel}(a)). \tag{35}$$

For each action class α, the boundary model must also declare an ordering function

$$Order(\alpha) : \mathsf{Cand}_\alpha(\alpha) \times \mathsf{Cand}_\alpha(\alpha) \times X \to \{-1, 0, 1\} \tag{36}$$

that induces, for each realised run under declared inputs x_{sel}, a total preorder together with a deterministic tie-break rule. If the realised run cannot induce a deterministic result using only declared inputs and declared ordering semantics, then the selected boundary is indeterminate and autonomous execution is non-admissible under default-deny semantics.

A compliant $EB(\alpha)$ artefact must therefore show that:

1. the candidate boundary set is finite and explicit,
2. the ordering relation is declared,
3. the earliest governed boundary can be selected deterministically under declared inputs,
4. the selected boundary is interceptable and non-bypass,
5. declared irreversible effects are causally dominated by the selected governed boundary.

In this paper, ''earliest'' is defined relative to the declared ordering semantics for the action class and the realised declared inputs. It does not mean absolute clock priority across arbitrary infrastructures. This directly links ISDAIRE to execution-boundary governance. If $EB(\alpha)$ is weak, OTANIS cannot rescue the class at runtime.

8 Conditions, Dependencies, and Version Binding

The seven named ISDAIRE elements are necessary but not operationally sufficient. A governed action class must also declare the concrete conditions and dependencies on which its admissibility and later runtime permission depend.

8.1 Condition Set

For each action class α, define a finite condition set

$$C(\alpha) = \{C_1(\alpha), \ldots, C_m(\alpha)\} \tag{37}$$

where each $C_i(\alpha)$ is a deterministic predicate over declared inputs, relevant state keys, and authority parameters.

For a realised execution state x at time t,

$$\mathsf{Cond}(\alpha, x, t) = \bigwedge_{i=1}^{m} C_i(\alpha, x, t). \tag{38}$$

The purpose of $C(\alpha)$ is to make the action class machine-legible at the point of enforcement. If a condition required for legitimacy exists only as an unwritten assumption, it is not part of the governed class.

8.2 Dependency Scope

Let

$$\mathsf{Dep}(\alpha) \subseteq \mathsf{Keys}(x) \tag{39}$$

be the declared dependency scope for action class α. This set must contain every state key on which $C(\alpha)$, lifecycle validity, revocation, boundary applicability, or any later runtime authority check functionally depends.

Let $\mathsf{Dep}^{\mathrm{inst}}(a)$ denote the realised instance-specific dependency scope after applying any declared parameterisation rules.

At runtime, the observed dependency set must match the declared realised scope:

$$\mathsf{Conform}(a, x, t) = 1 \Leftrightarrow \mathsf{ObservedDep}(a) = \mathsf{Dep}^{\mathrm{inst}}(a). \tag{40}$$

This requirement matters because hidden dependencies create a false impression of governed determinism. If governance-relevant state enters through undeclared caches, fallbacks, or derived features, the class was not specified completely [15].

8.3 Version Binding

Let $V(\alpha)$ denote the version and integrity metadata associated with the governed specification object for class α. At minimum, version binding must preserve the following boundary-governance tuple

$$R = \{(\alpha, \mathsf{Cand}_\alpha(\alpha), \mathsf{SelectEarliest}_\alpha, Order(\alpha), \mathsf{Dep}(\alpha), IA(\alpha), v)\} \tag{41}$$

where $v \in \mathbb{N}$ is a monotonically increasing approved governance version.

Accordingly, $V(\alpha)$ must bind at least:

1. the action-class definition,
2. the condition set $C(\alpha)$,
3. the dependency scope $\mathsf{Dep}(\alpha)$,
4. the candidate boundary set $\mathsf{Cand}_\alpha(\alpha)$,
5. the selection contract $\mathsf{SelectEarliest}_\alpha$,
6. the ordering model $Order(\alpha)$,
7. the authority-source binding and approval state.

At runtime, a class can only claim strict governance if the executing artefacts match the approved governed-specification version:

$$\mathsf{VersionMatch}(a) = 1 \Leftrightarrow v_{\mathrm{runtime}}(a) = v_G(\alpha(a)) \tag{42}$$

where $v_G(\alpha)$ denotes the approved governed-specification version for action class α.

If the runtime artefacts, boundary-selection logic, or dependency declarations do not match the approved version, the class may remain documented but it is no longer strictly governed in the OTANIS sense.

Element	Governing question	Minimum governed artefact	Why runtime depends on it
$I(\alpha)$	What is this class for	Purpose definition and allowed intent labels	Prevents broad or ambiguous action semantics
$S(\alpha)$	What state transition is allowed	Permitted effect envelope	Constrains the allowed post-state
$DS(\alpha)$	Where must this class never operate	Forbidden regions, exclusions, protected classes	Stops silent crossing into barred domains
$AS(\alpha)$	Who may authorise it	Principals, issuers, delegation semantics	Makes authority attributable and bounded
$IA(\alpha)$	What irreversible effects are at stake	Irreversibility classification and declared effect set	Determines whether boundary control is required
$RF(\alpha)$	Under which risk frame is it acceptable	Thresholds, escalation rules, refusal conditions	Prevents implicit or ad hoc risk semantics
$EB(\alpha)$	Where must governance bind at runtime	Candidate boundaries and deterministic binding rule	Enables execution-boundary enforcement
$C(\alpha)$	Which concrete predicates must hold	Deterministic rule set	Gives the gate something exact to evaluate
$\mathsf{Dep}(\alpha)$	Which state does legitimacy depend on	Declared dependency scope	Prevents hidden inputs from shaping execution
$V(\alpha)$	Which approved artefacts are in force	Version and integrity bindings	Makes change control auditable

Table 1: Minimum governed artefacts required to make an action class reviewable for canonical ISDAIRE admissibility and for the stronger autonomous-deployment qualification developed in this paper. The table summarises required artefacts. It does not by itself establish semantic adequacy or deployment acceptability.

9 A Compact Summary of the Specification Obligations

10 Completeness, Adequacy, and Design-Time Failure

10.1 Syntactic Completeness

Define syntactic completeness relative to the declared specification object:

$$\mathsf{Complete}(G(\alpha)) = 1 \Leftrightarrow \mathsf{WellFormed}(G(\alpha)) \wedge \mathsf{BoundaryComplete}(\alpha) \wedge \mathrm{Declared}(\mathsf{Dep}(\alpha)). \quad (43)$$

More concretely, syntactic completeness requires that all required artefacts exist in an explicit, typed, versioned form. It is decidable relative to the schema and integrity rules of the deployment.

10.2 Dependency Completeness

Dependency completeness requires that the declared dependency scope is sufficient for the predicates the governance gate will later evaluate. It is partially testable through runtime observation:

$$\mathsf{ObservedDep}(a) = \mathsf{Dep}^{\mathrm{inst}}(a) \quad (44)$$

for all governed instances a.

However, that equality only proves gate-level dependency closure. It does not prove that every real-world hazard was represented in the class. An omitted fraud source or omitted legal-hold register may remain invisible until the omitted circumstance arises.

10.3 Boundary Completeness

Boundary completeness is especially important for execution-bearing classes. Let $IE_{decl}(\alpha)$ be the declared set of irreversible external effects and $\mathsf{Cand}_\alpha(\alpha)$ the declared candidate boundary set. Then:

$$\mathsf{BoundaryComplete}(\alpha) = 1 \quad (45)$$

iff every declared irreversible effect is associated with a candidate boundary structure that can be deterministically governed, and no known or plausibly binding precursor has been excluded without accountable justification.

For high-consequence execution-bearing action classes, boundary completeness requires more than a declarative statement. At minimum it requires:

1. mandatory review of precursor categories including reservation or allocation calls, queue admissions, mirrored or replicated writes, staged dispatch artefacts, third-party acceptances, retries, and compensating paths,
2. at least one explicitly accountable challenger review function whose role is to challenge precursor classification and boundary dominance claims rather than merely to approve the authored model,
3. empirical workflow tracing, reconciliation evidence, staging traces, or equivalent operational evidence sufficient to test whether supposedly preparatory transitions already create externally relevant commitment,
4. re-certification whenever topology, integration semantics, third-party commit behaviour, or workflow staging changes materially,
5. non-admissibility whenever precursor semantics remain unresolved, contested, or only weakly evidenced.

This is not guaranteed by runtime enforcement. It depends on correct ex-ante boundary discovery and review [6, 14]. If a real precursor commitment is semantically laundered out of $IE_{decl}(\alpha)$ or $\mathsf{Cand}_\alpha(\alpha)$, the class may remain structurally coherent on paper while being dangerously weak in reality.

10.4 Semantic Adequacy

Semantic adequacy is a different question. Define

$$\mathsf{Adequate}(\alpha, D) \tag{46}$$

as the property that the governed specification for action class α is actually sufficient for the domain D.

This property is not decidable inside the formal system. It requires external judgement, domain-owner review, adversarial testing, and sometimes legal or regulatory interpretation [3, 14]. The framework makes that limitation explicit because ignoring it produces false confidence.

10.5 Interpretive Consequence

A deployment may therefore satisfy

$$\mathsf{AdmissibleISDAIRE}(\alpha) = 1$$

and still be semantically weak. That does not make the framework defective. It means the framework is honest about the line between formal admissibility and domain sufficiency. Runtime governance can only enforce declared semantics. It cannot manufacture missing domain truth.

11 Systematic Construction Method

This section states the minimum governed process required to synthesise an action class whose specification object is structurally sound, canonically admissible under ISDAIRE, and, where appropriate, strong enough to support the stricter autonomous-deployment qualification.

11.1 Step 1, Action-Class Decomposition

The deployment must first decompose high-level autonomous behaviour into discrete action classes $\alpha \in \mathcal{A}_c$. A class is too broad if materially different authority, risk, or boundary semantics are hidden inside a single label. It is too narrow if it fragments one real action class into arbitrary implementation details that obscure responsibility.

11.2 Step 2, Intent and Effect Definition

For each class, the deployment must define:

1. the permitted intent family $I(\alpha)$,
2. the allowed effect-transition envelope $S(\alpha)$,
3. the forbidden regions $DS(\alpha)$.

These three artefacts together define what the class is, what it can do, and where it must not operate.

11.3 Step 3, Authority Modelling

The deployment must define $AS(\alpha)$, including any acceptable issuers, principals, roles, delegation chains, approval surfaces, and expiry assumptions. Authority must be bounded to the class. A principal with authority to approve a claim is not automatically a valid source for direct settlement submission unless that relation is explicitly defined.

11.4 Step 4, Irreversibility and Boundary Discovery

The deployment must identify:

1. declared irreversible effects $IE_{decl}(\alpha)$,
2. the candidate boundary set $\mathsf{Cand}_\alpha(\alpha)$,
3. the ordering model $Order(\alpha)$,
4. the deterministic selection rule $\mathsf{SelectEarliest}_\alpha$.

This step must consider not only obvious final commits but also precursor transitions that may already bind capacity, funds, legal exposure, compliance state, inventory state, queue admission, or external expectations. For high-consequence systems, precursor boundary review is not optional.

A compliant construction process for this step must include:

1. explicit precursor-category review,
2. documented rationale for any precursor exclusion,
3. accountable domain-owner approval for any claim that a precursor is merely preparatory,
4. challenger review of the declared earliest governed boundary,
5. empirical workflow tracing or reconciliation evidence where architecture alone is insufficient to establish boundary dominance.

If the precursor semantics of a transition remain unresolved, then $\mathsf{AutonomousAdmissible}(\alpha) = 0$ for that action class. In that case the correct response is redesign, stronger evidence, or non-admissibility, not optimistic declaration.

11.5 Step 5, Risk Framing

The deployment must define $RF(\alpha)$, including risk thresholds, escalation semantics, refusal conditions, and any special control requirements tied to the class. A class whose risk semantics remain implicit is not mature enough for governed automation.

11.6 Step 6, Predicate and Dependency Construction

The deployment must define deterministic conditions $C(\alpha)$ and a declared dependency scope $\mathsf{Dep}(\alpha)$. Where payload-dependent state is needed, the parameterisation rule itself must be declared and version bound.

11.7 Step 7, Review, Approval, and Version Binding

Finally, the specification object must pass explicit governance review:

$$\mathsf{GovernanceReview}(G(\alpha)) = 1 \tag{47}$$

only if the approved authority source has reviewed, versioned, and integrity-bound the artefacts.

This step is not administrative decoration. Without it, the specification is not actually governed. It is merely documented.

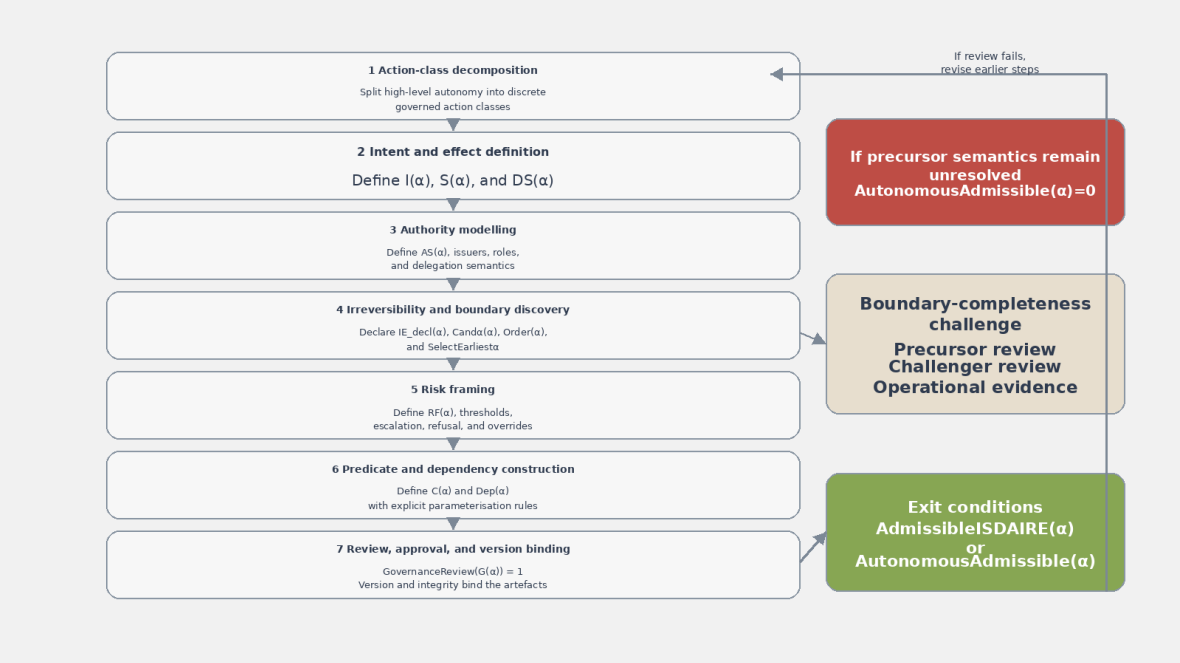

Figure 3: Systematic construction method for a governed action class under ISDAIRE. The sequence moves from action-class decomposition through authority, boundary, risk, predicate, and dependency construction into review and version binding. If precursor semantics remain unresolved, autonomous admissibility fails even when the specification object is otherwise structurally present.

12 Worked Example, Insurance Claim Settlement

12.1 Domain

Consider an insurance claim-settlement system that may autonomously submit certain low-risk approved claims to a payment rail once defined conditions hold. This is a suitable example because claim settlement is materially execution-bearing. Once submission to the correct irreversible boundary occurs, the organisation may face financial, legal, and operational consequences that cannot be dismissed as mere model output.

12.2 Action Class

Define the action class

$$\alpha = \mathsf{ClaimSettlement}. \tag{48}$$

12.3 Intent

The governed intent family is:

$$I(\alpha) = \{\mathsf{ValidClaimPayment}\}. \tag{49}$$

This means the class exists only to settle a valid claim that has already passed required prior steps. It does not include goodwill payments, legal settlements outside policy scope, recovery

disbursements, or compensation for disputed claims.

12.4 Scope

Define the permitted effect envelope:

$$S(\alpha) = \{(\mathsf{ApprovedClaim}, \mathsf{PaidClaim})\}. \tag{50}$$

This explicitly excludes transitions from UnderInvestigation, LegalHold, FraudFlagged, or equivalent states into PaidClaim.

12.5 Domain Separation

Define

$$DS(\alpha) = \{\mathsf{FraudFlagged}, \mathsf{LegalHold}, \mathsf{SanctionedCounterparty}, \mathsf{JurisdictionBlocked}\}. \tag{51}$$

These regions may not be crossed by this class, regardless of what an upstream model suggests.

12.6 Authority Source

Let

$$AS(\alpha) = \{\mathsf{ClaimsManager}, \mathsf{DelegatedSettlementService}\}. \tag{52}$$

The presence of DelegatedSettlementService is only valid if its delegation chain is formally bound to a recognised principal and to this specific action class. A generic service token is insufficient [10, 11].

12.7 Irreversibility Awareness

Suppose the deployment declares:

$$r(\alpha) = 1 \tag{53}$$

and

$$IE_{decl}(\alpha) = \{\mathsf{PaymentRailSubmission}\}. \tag{54}$$

The candidate boundary set is

$$\mathsf{Cand}_{\alpha}(\alpha) = \{\mathsf{ERPCommit}, \mathsf{PaymentRailSubmission}\}. \tag{55}$$

Let the declared ordering model $Order(\alpha)$ rank candidate boundaries by earliest governed irreversible commitment under the realised settlement path and declared routing inputs $x_{sel}(a)$. Then the bound boundary is

$$T_e^*(a) = \mathsf{SelectEarliest}(\mathsf{Cand}(a), x_{sel}(a)). \tag{56}$$

If the ERP commit is locally reversible and does not itself guarantee external settlement, then for the realised run one may have

$$T_e^*(a) = \mathsf{PaymentRailSubmission}. \tag{57}$$

If, however, the ERP commit already creates an irreversible queue admission, settlement reservation, or other economically binding precursor that cannot be cancelled reliably, then the

declared candidate set is incomplete or misordered and the class fails the stronger autonomous qualification unless the earlier event is explicitly governed.

12.8 Risk Framing

Define:

$$RF(\alpha) = (\mu_\alpha, \tau_\alpha, \varepsilon_\alpha, \kappa_\alpha, \phi_\alpha) \tag{58}$$

where μ_α includes amount, sanctions exposure, anomaly score, and claims-history variance, and ϕ_α records the consequence-class, freshness, and override assumptions under which the frame is to be interpreted. For illustration:

$$\mu_\alpha(x) = (\mathsf{ClaimAmount}(x), \mathsf{AnomalyScore}(x)). \tag{59}$$

Low-value claims below a threshold may remain auto-settle eligible. Higher-risk claims must escalate.

12.9 Condition Set

A minimal condition set could be:

$$C_1(\alpha) := \mathsf{ApprovedClaim} \tag{60}$$
$$C_2(\alpha) := \mathsf{NoFraudFlag} \tag{61}$$
$$C_3(\alpha) := \mathsf{AuthorityValid} \tag{62}$$
$$C_4(\alpha) := \mathsf{SanctionsClear} \tag{63}$$
$$C_5(\alpha) := \mathsf{BankDetailsVerified}. \tag{64}$$

Then

$$\mathsf{Cond}(\alpha, x, t) = \bigwedge_{i=1}^{5} C_i(\alpha, x, t). \tag{65}$$

12.10 Dependencies

Define

$$\begin{aligned}\mathsf{Dep}(\alpha) = \{&\texttt{claim\_status}, \texttt{fraud\_registry}, \texttt{authority\_registry},\\ &\texttt{sanctions\_service}, \texttt{beneficiary\_verification}\}.\end{aligned} \tag{66}$$

If the real gate also relies on a cached anomaly model, a policy-routing table, or an undeclared fallback account mapping, then the class was incompletely specified even if the declared rules look plausible.

12.11 Failure Cases Exposed by ISDAIRE

The example makes the design value of ISDAIRE concrete.

Failure F1, Omitted dependency. If `fraud_registry` is omitted from $\mathsf{Dep}(\alpha)$, runtime enforcement may still work correctly relative to the declared class while the class itself remains weak.

Failure F2, Wrong boundary. If ERPCommit already creates an irreversible external queue admission that cannot be cancelled reliably, then $EB(\alpha)$ was mis-specified.

Failure F3, Overbroad authority. If DelegatedSettlementService can also pay ex gratia claims or cross-jurisdiction settlements, then $AS(\alpha)$ is too broad for this class.

Failure F4, Weak scope. If the scope artefact treats all approved claims alike and ignores pending recovery or litigation status, then $S(\alpha)$ is semantically weak.

Failure F5, stale risk frame. If sanctions or beneficiary verification freshness is not bound into the class, the system may act on outdated evidence.

In each case, the failure originates in the ex-ante specification layer. OTANIS can later refuse a runtime instance when evidence is stale or dependencies drift, but it cannot repair a class whose business semantics were never defined tightly enough.

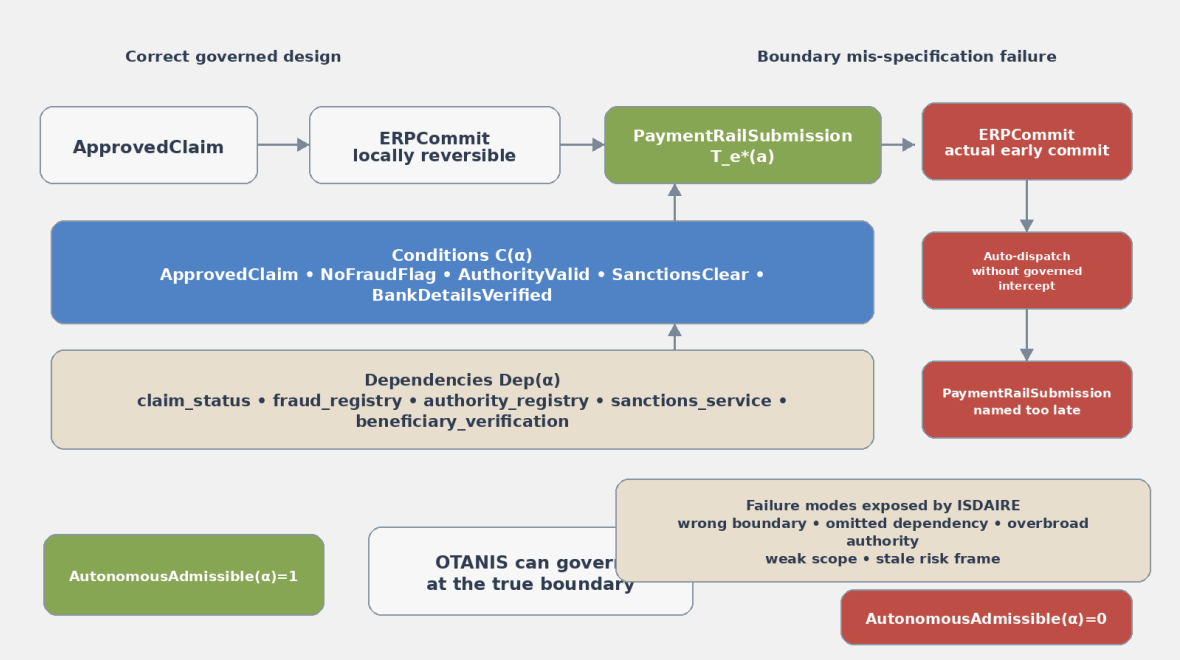

Figure 4: Insurance claim-settlement example. The left side shows a correctly governed design in which claim conditions and declared dependencies support governance at the true boundary $T_e^*(a)$. The right side shows boundary mis-specification, where an earlier effective commitment invalidates autonomous admissibility even though a later payment step was named as the boundary.

13 Relationship to OTANIS and MGAG

ISDAIRE is the ex-ante admissibility layer. OTANIS is the runtime execution-governance layer. MGAG is the compositional preservation layer across multiple domains or governance strata [1, 2].

Their relationship should be expressed in two layers rather than collapsed into one. In the equations below, $\mathsf{Permit}(\cdot)$, $\beta_e(a)$, $x_e(a)$, $T_e^*(a)$, and $\mathsf{Permit}^{(k)}(\cdot)$ are runtime symbols inherited from the OTANIS layer and are used here by reference rather than redefined in full inside this paper.

First, OTANIS runtime permission presupposes the canonical class-level admissibility predicate:

$$\mathsf{Permit}(a, \beta_e(a), x_e(a)) \Rightarrow \mathsf{AdmissibleISDAIRE}(\alpha(a)). \tag{67}$$

Second, this paper introduces a stronger deployment-side qualification for autonomous execution:

$$\mathsf{AutonomousAdmissible}(\alpha) = 1 \Leftrightarrow \mathsf{AdmissibleISDAIRE}(\alpha) \wedge \mathsf{BoundaryComplete}(\alpha). \tag{68}$$

Accordingly,

$$\mathsf{AutonomousAdmissible}(\alpha) = 1 \Rightarrow \mathsf{AdmissibleISDAIRE}(\alpha) = 1,$$

but not conversely. A class may satisfy the canonical OTANIS admissibility predicate while still failing the stronger qualification used in this paper because boundary completeness is weak or precursor semantics remain unresolved. Even where $\mathsf{AutonomousAdmissible}(\alpha) = 1$ holds, a realised instance may still be refused at runtime because evidence is stale, authority has been revoked, the boundary is indeterminate, or the realised dependencies do not conform.

In multi-layer settings, admissibility must survive layered governance rather than merely local checks. Let $k \in \{1, \ldots, L\}$ index governance layers. Then one may write

$$MGAG(a) = \bigwedge_{k=1}^{L} \mathsf{Permit}^{(k)}(a, \beta^{(k)}, x_e(a), T_e^*(a)). \tag{69}$$

For such a conjunction to be meaningful, each relevant action class must already satisfy the canonical ex-ante admissibility predicate $\mathsf{AdmissibleISDAIRE}(\alpha)$. Otherwise multi-layer runtime enforcement only composes weak local artefacts.

14 Falsifiability and Review Criteria

A framework is only useful if it can be failed. ISDAIRE therefore admits clear falsification conditions.

F1, Artefact absence. There exists an action class α for which one or more governed artefacts required for $I(\alpha), S(\alpha), DS(\alpha), AS(\alpha), IA(\alpha), RF(\alpha)$, or $EB(\alpha)$ is missing, malformed, unverifiable, or not machine-legible where required, yet the deployment claims $\mathsf{AdmissibleISDAIRE}(\alpha) = 1$.

F2, Inconsistency. There exists an action class α where $S(\alpha)$ and $DS(\alpha)$, or $IA(\alpha)$ and $EB(\alpha)$, are materially contradictory, yet the class is still treated as admissible.

F3, Boundary non-dominance. There exists an action class α whose declared boundary does not causally dominate a declared irreversible effect, or whose commit path is bypassable, yet $EB(\alpha) = 1$ is asserted.

F4, Hidden dependencies. There exists a governed action instance a such that

$$\mathsf{ObservedDep}(a) \neq \mathsf{Dep}^{\mathrm{inst}}(a)$$

for governance-relevant evaluation, yet the deployment claims the class was completely specified.

F5, Authority-source mismatch. There exists an executed or execution-eligible class where the real approving or issuing source falls outside $AS(\alpha)$.

F6, Version drift. There exists an action instance a where runtime artefacts do not match the approved governed specification version for $\alpha(a)$, yet the class is still presented as strictly governed.

These falsifiers matter because they make the framework auditable. A reviewer does not need to agree with every philosophical view of AI governance to test whether a deployment satisfies the

stated admissibility conditions.

15 Implications for Design Review and Procurement

ISDAIRE has direct use beyond academic formalism.

For design review, it gives engineers a checklist of formal obligations that must be satisfied before runtime governance claims are credible. For procurement, it provides a way to distinguish between systems that say they have policy controls and systems that can show governed action classes with explicit boundaries, dependencies, authority sources, and versioned artefacts. For audit, it helps separate failures of semantic construction from failures of runtime enforcement.

This is especially useful in agentic systems, because vendors may claim strong control based on monitoring, approval workflows, or broad access-control language while leaving the action-class semantics under-specified. ISDAIRE gives a more precise question to ask. Which execution-bearing classes are canonically admissible under the declared artefacts, which are only structurally sound as governed specification objects, under which authority sources and dependency scopes do they operate, at which governed boundaries are they bound, and under which approved version do those claims hold.

15.1 Restriction on Procurement and Assurance Claims

A claim that $\mathsf{StructuralAdmissible}(\alpha) = 1$ means only that the governed specification object for action class α is well formed, internally consistent, and actually governed. A claim that $\mathsf{AdmissibleISDAIRE}(\alpha) = 1$ means that the canonical OTANIS ex-ante admissibility contract has been satisfied for the declared artefacts, including $I(\alpha)$, $S(\alpha)$, $DS(\alpha)$, $AS(\alpha)$, $IA(\alpha)$, $RF(\alpha)$, $EB(\alpha)$, and the irreversibility-consistency condition. Neither claim, by itself, means that the class is semantically adequate, safe, fair, legally sufficient, regulatorily acceptable, or fit for autonomous deployment in the relevant domain. Where the stronger claim concerns autonomous execution readiness under OTANIS-compatible runtime control, it must instead be stated at the level of $\mathsf{AutonomousAdmissible}(\alpha) = 1$, with its stricter boundary conditions made explicit.

Accordingly:

1. any assurance or procurement claim under ISDAIRE must be scoped to the specific action class or bounded class family to which it applies,
2. neither structural admissibility nor canonical ISDAIRE admissibility must be marketed or represented as equivalent to system safety, domain validity, or regulatory compliance,
3. where semantic adequacy, legal sufficiency, or operational deployability have not been independently established, that absence must be stated explicitly rather than implied away by the presence of governed artefacts.

This restriction is not rhetorical caution. It is necessary because structurally strong specification can coexist with semantically wrong domain assumptions.

16 Discussion

A critical property of the framework is that the execution boundary is not assumed to be a single fixed point known independently of the realised run. Instead, ISDAIRE defines the candidate boundary space and the declared rules for binding the governed boundary at runtime. The actual governed boundary $T_e^*(a)$ is selected deterministically from $\mathsf{Cand}(a)$ under declared selection inputs and declared ordering semantics. This is exactly why the framework can remain

consistent with OTANIS across heterogeneous execution paths. Governance is attached to the earliest declared and governable commitment point for the realised run, not to a pre-assumed step in an abstract workflow.

The strongest objection to a framework like ISDAIRE is that formal specification alone cannot capture every real-world hazard. That objection is correct, and the paper does not try to evade it. No ex-ante formalism can guarantee semantic adequacy for all future domain circumstances. Safety engineering, law, organisational judgement, and operational change all exceed what a single action-class schema can settle [14].

But that objection does not weaken the present contribution. It clarifies its boundary. If runtime governance is to be meaningful, then the action class must first be represented as a governed specification object. That requirement is not made false by the fact that semantic adequacy remains a harder problem. It becomes more important.

A second objection is that existing access-control or zero-trust frameworks already address this space. They do address part of it [11,16]. However, they do not by themselves provide a complete ex-ante admissibility architecture for execution-bearing action classes tied to irreversibility semantics, declared boundary binding, dependency completeness, and OTANIS-compatible runtime governance. The point here is not to replace those frameworks but to integrate their relevant primitives into a specification discipline suited to irreversible agentic execution.

A third objection is practical. The specification effort may be large. That is also correct. In high-consequence settings, the effort should be large. The cost of weak ex-ante semantics is merely hidden until the system reaches scale, composition, or irreversibility.

17 Conclusion

This paper introduced ISDAIRE as a formal specification and admissibility framework for execution-bearing action classes in agentic AI systems. The framework is intentionally narrower than a universal theory of AI governance. It does not claim to guarantee semantic adequacy, safety, legal sufficiency, or domain correctness. It does not discover omitted hazards automatically or repair poorly designed action classes at runtime. What it does claim is more precise.

It claims that executable governance for irreversible agentic action requires an ex-ante governed specification object that defines intent, scope, domain separation, authority source, irreversibility awareness, risk framing, execution boundary, concrete condition sets, dependency scope, and version binding. It further claims that the canonical class-level admissibility predicate $\mathsf{AdmissibleISDAIRE}(\alpha)$ remains the correct ex-ante OTANIS-compatible test of whether the declared governance contract exists and is internally coherent for the action class. The paper also introduces $\mathsf{StructuralAdmissible}(\alpha)$ only as a narrower auxiliary predicate for specification-object soundness, and $\mathsf{AutonomousAdmissible}(\alpha)$ as the stronger deployment-side qualification used when boundary completeness is additionally required.

That claim is useful because it makes a previously blurred boundary explicit. OTANIS can enforce at runtime only what ISDAIRE has specified coherently ex ante. If the class is weak, runtime governance remains bounded by that weakness. If the class satisfies the canonical admissibility contract and the stronger boundary-completeness conditions, runtime governance becomes materially more meaningful, auditable, and defensible. But neither canonical admissibility nor autonomous qualification is the same thing as semantic adequacy. A class may therefore be formally admissible and still be semantically wrong at the same time.

Read in that strict sense, ISDAIRE is not a correction to OTANIS but its formal prerequisite

layer. It provides the specification discipline required before execution-boundary governance can be asserted without hand-waving. That is the real contribution of the framework. It moves admissibility from intuition and prose toward governed formal structure, while stating plainly that semantic truth remains an external and irreducible burden on design, review, and domain judgement.

References

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

[2] ------, ‘‘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.

[3] 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

[4] ------, ‘‘Artificial intelligence risk management framework: Generative artificial intelligence profile,’’ NIST, Tech. Rep. NIST AI 600-1, 2024. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

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

[6] 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.

[7] F. B. Schneider, ‘‘Enforceable security policies,’’ *ACM Transactions on Information and System Security*, vol. 3, no. 1, pp. 30--50, 2000.

[8] C. A. R. Hoare, ‘‘An axiomatic basis for computer programming,’’ *Communications of the ACM*, vol. 12, no. 10, pp. 576--580, 1969.

[9] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999.

[10] 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.

[11] 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

[12] 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.

[13] 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.

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

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

[16] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, ‘‘Zero trust architecture,’’ NIST, Tech. Rep. NIST Special Publication 800-207, 2020. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-207.pdf