SDPF vNext Core Specification

Specification-Driven Governance and Execution Framework for Stochastic AI Systems

Integrated Architecture Draft — 2026-05-08

Abstract

SDPF defines a specification-centric governance and execution framework for stochastic AI systems. The framework establishes deterministic operational governance around probabilistic generation systems through formal specifications, invariant extraction, constraint enforcement, verification gates, provenance tracking, and closed-loop execution control.

The framework does not claim deterministic AI generation. Instead, it constrains admissible outputs within formally governed acceptance boundaries. SDPF therefore shifts reliability from exact artifact replication toward invariant-preserving conformance.

This document defines the architectural model, execution semantics, invariant system, verification lifecycle, runtime governance model, and future formalization direction for SDPF vNext.

1. Problem Definition

Modern AI systems violate a foundational assumption of traditional software engineering: deterministic execution.

Given identical prompts, stochastic models may produce:
- different implementations,
- different internal structures,
- different execution flows,
- and different operational behavior.

Traditional software assurance methods rely heavily on canonical reproducibility. AI-native systems require a different operational model.

The governing problem becomes:

How can probabilistic generators be operationally governed through deterministic verification and policy boundaries?

SDPF addresses this through specification-defined admissibility constraints.

2. Core Principle

SDPF is founded on the following principle:

stochastic generation may vary internally while operational governance remains deterministic externally.

The framework therefore governs:
- admissibility,
- deployment eligibility,
- verification status,
- runtime authorization,
- provenance,
- and policy conformance.

The system does not require identical outputs. It requires preservation of formally defined invariants.

3. Formal Definitions

Specification S

A complete executable governance contract.

Generated Artifact G

Any generated implementation, configuration, workflow, or executable artifact.

Invariant Set I(S)

The full set of machine-verifiable constraints derived from S.

Conformance

$Conforms(G,S)$

Invariant Satisfaction

$Sat(G,I(S))$

Specification Completeness

$Comp(S)$

Constraint Satisfiability

$SAT(I(S))$

Formal framing:

$$Conforms(G,S) \equiv Comp(S) \land Sat(G,I(S))$$

For complete specifications:

$$\text{Conforms}(G,S) \leftrightarrow \text{Sat}(G,I(S))$$

Constraint satisfiability requirement:

$$\exists G : \text{Sat}(G,I(S))$$

Interpretation:

A specification is operationally admissible only if all required invariants can be satisfied simultaneously.

4. Bounded Stochasticity

Bounded stochasticity does not mean deterministic generation.

It means:

$$\text{stochastic variation constrained inside invariant-preserving admissibility boundaries.}$$

Acceptable outputs may differ in:
- formatting,
- structure,
- implementation details,
- naming,
- execution decomposition,

while preserving:
- required behavior,
- security guarantees,
- interface contracts,
- policy requirements,
- and runtime constraints.

The acceptance boundary becomes deterministic even if generation remains probabilistic.

5. Invariant System

SDPF treats invariants as first-class governance objects.

Invariant classes include:

Structural Invariants

File structure, schema validity, dependency existence.

Behavioral Invariants
Functional tests, execution behavior, contract compliance.

Security Invariants
Access controls, forbidden operations, sandbox boundaries.

Temporal Invariants
Retry limits, timeout guarantees, sequencing constraints.

Resource Invariants
CPU, memory, latency, throughput limits.

Semantic Invariants
Observable behavior preservation.

Policy Invariants
Regulatory, organizational, and deployment rules.

All generated artifacts must satisfy all mandatory invariant classes.

6. Invariant Algebra

SDPF vNext introduces invariant composition semantics.

Composition:
$I_{total} = I_a \cup I_b \cup I_c$

Refinement:
stronger invariants may refine weaker invariants.

Inheritance:
parent specifications propagate invariant subsets.

Conflict Detection:
incompatible invariants invalidate satisfiability.

Priority Semantics:
mandatory invariants override advisory invariants.

Runtime Propagation:
invariant modifications propagate through dependent artifacts.

The invariant system therefore functions as an executable governance graph rather than a static checklist.

7. State Machine Semantics

SDPF defines governed execution as a lifecycle state machine.

Canonical lifecycle:

Draft
→ Parsed
→ Validated
→ Constraint-Solved
→ Generated
→ Verified
→ Proven
→ Deployable
→ Runtime-Observed
→ Closed

Every transition requires:
- invariant preservation,
- provenance evidence,
- policy authorization,
- transition guards,
- and verification artifacts.

Example:

Generated → Verified
Guard:
  all structural and behavioral invariants satisfied

Verified → Deployable
Guard:
  security, policy, and runtime constraints satisfied

8. Canonical Intermediate Representation

A canonical semantic IR forms the computational core of SDPF.

Execution pipeline:

SDPF Specification

↓
Parser
↓
Semantic IR
↓
Invariant Extraction
↓
Constraint Graph
↓
Generation Runtime
↓
Verification Engine
↓
Closure Record

The IR serves as:
- the orchestration substrate,
- the invariant attachment layer,
- the verification target,
- the dependency graph anchor,
- and the execution coordination model.

Without a canonical IR, SDPF remains document-oriented instead of computationally executable.

9. Closed-Loop Execution Runtime

SDPF vNext transitions from static validation into adaptive governed execution.

Execution cycle:

Specification
→ Generation
→ Verification
→ Failure Analysis
→ Constraint Refinement
→ Regeneration
→ Reverification
→ Deployment Authorization

Verification therefore becomes part of runtime orchestration rather than post-generation inspection.

This architecture enables:
- adaptive correction,

- autonomous remediation,
- bounded regeneration,
- and governed execution loops.

10. Verification Hierarchy

Verification assurance levels:

L0 — syntax validity
L1 — structural completeness
L2 — behavioral correctness
L3 — security validation
L4 — semantic equivalence confidence
L5 — runtime invariant preservation

These levels define progressively stronger operational assurance guarantees.

11. Runtime Governance

Governance persists after deployment.

Runtime governance capabilities include:
- invariant drift detection,
- runtime policy enforcement,
- deployment revocation,
- telemetry analysis,
- rollback triggering,
- continuous verification,
- and provenance updates.

SDPF therefore governs the entire operational lifecycle rather than only pre-deployment validation.

12. Provenance Architecture

Every governed transition emits evidence artifacts.

Required provenance elements:
- signatures,
- timestamps,
- invariant results,
- dependency lineage,
- generation metadata,
- verification records,

- runtime telemetry,
- and closure status.

This creates auditable execution lineage suitable for regulated operational environments.

13. Probabilistic Containment

Future SDPF formalization extends bounded stochasticity into measurable probabilistic containment.

Target form:

$$P(G \notin C(S)) \leq \varepsilon$$

Where:
G = generated artifact
$C(S)$ = admissible conformance region
ε = bounded failure probability

This introduces:
- convergence analysis,
- entropy reduction,
- measurable admissibility,
- and probabilistic governance guarantees.

14. Semantic Correctness

Structural correctness does not guarantee semantic correctness.

A generated artifact may:
- compile,
- pass tests,
- satisfy structural invariants,
- and satisfy policy constraints,

while still implementing incorrect business logic.

Future SDPF research therefore includes:
- observational equivalence,
- symbolic execution,
- semantic invariant systems,
- executable semantics,
- and behavioral correctness analysis.

15. Enterprise Execution Architecture

SDPF is designed for regulated and high-assurance environments requiring:
- auditability,
- traceability,
- deployment governance,
- evidence retention,
- compliance verification,
- and operational accountability.

Target deployment domains include:
- healthcare,
- finance,
- aerospace,
- defense,
- industrial automation,
- and enterprise AI infrastructure.

16. Strategic Positioning

SDPF should be positioned as:

> deterministic governance infrastructure
> for stochastic execution systems

not as:
> deterministic AI.

The framework governs operational admissibility rather than attempting to eliminate probabilistic generation.

17. Long-Term Direction

SDPF is converging toward:

- governed autonomous execution,
- proof-carrying generation,
- invariant-aware orchestration,
- continuous runtime verification,
- and specification-driven AI operating infrastructure.

The long-term architecture resembles a governance kernel for AI-native execution systems.

18. Development Roadmap

Phase 1 — Formal Semantics
- constraint logic
- invariant algebra
- transition semantics
- satisfiability theory

Phase 2 — Execution Infrastructure
- semantic IR
- constraint engine
- verification runtime
- provenance ledger

Phase 3 — Autonomous Governance
- adaptive remediation
- runtime correction loops
- continuous governance

Phase 4 — Semantic Intelligence
- symbolic reasoning
- semantic equivalence
- behavioral verification
- proof-aware execution

19. Final Statement

SDPF defines a governed execution architecture for stochastic AI systems.

The framework's core innovation is not deterministic generation. The core innovation is deterministic operational governance around probabilistic generation systems through invariant-centered admissibility, verification-aware execution, and lifecycle-governed orchestration.

The architecture ultimately positions specifications as executable governance contracts for AI-native systems.