THE RAINFALL COHERENCE SYSTEM (RCS)

The runtime for building Coherent Applications:
Deterministic Governance over Probabilistic Generative Substrates

Rainfall Inc. · Updated on June 23rd, 2026

Abstract

Generative models are powerful but probabilistic: given the same situation they may decide differently across prompts, model versions, vendors, and time. A useful answer is not yet a dependable service, and a completed task is not yet a trusted product — and for autonomous agents that take actions in the world, this variance — not raw capability — is the barrier to production deployment wherever consistency, accountability, and trust are prerequisites. The *Rainfall Coherence System (RCS)* is the runtime for building **Coherent Applications (CApps)**: governed agentic applications that turn probabilistic intelligence into a service that behaves dependably across models, vendors, and time. It addresses the variance gap directly. RCS treats coherence — the same scenario yielding the same governed decision regardless of the underlying model or substrate — as a structural property of a separate control layer rather than of any individual model or prompt. It supervises agents at their observable surfaces (context, specification, model, and tool-call boundaries) without altering their intrinsic behavior; it situates each emission within a combined behavioral-and-context manifold and enforces conformance against learned *imprints*; it executes a fixed per-step reflex chain that gates emissions before and after they occur, meters cost, and anchors a tamper-evident audit trail; and it functions as a substrate-neutral meta-orchestrator that runs agentic workflows directly or hands them off to external engines. The runtime is decentralized and accelerator-free, executing from cloud clusters down to edge devices. The CApp it produces packages capability (what it does), behavioral continuity (how it stays itself over time), and sovereign presence (who it represents and how it shows up) into one governed, portable artifact. This paper documents the problem, the architecture, the coherence mechanism, and an evaluation methodology, and situates RCS against orchestration frameworks, agent platforms, governance tooling, and durable-execution engines.

Keywords: agentic AI, behavioral governance, coherence, orchestration, runtime determinism, model routing, audit, edge inference.

1. Introduction

Artificial intelligence has crossed a threshold. The last decade produced systems that can perceive and reason; the current one is producing systems that can act — software that pursues goals, invokes tools, and carries out multi-step work with limited human supervision. That shift turns a capability question into an operational one: not whether a model is intelligent enough, but whether the systems built on top of it behave dependably enough to be trusted with real responsibility. This paper introduces the Rainfall Coherence System (RCS), the runtime that answers that question by producing a new kind of

© 2026 Rainfall Inc

deliverable — the Coherent Application — and develops the argument that the durable value of the agentic era accrues not to the models that generate intelligence, but to the layer that turns that intelligence into applications whose behavior can be trusted.

The deployment of large language models has moved from generating text to *taking action*. Agents now select tools, call APIs, route work, escalate, refuse, and approve. As they do, the failure mode that matters shifts from a system *saying* the wrong thing to a system *doing* the wrong thing — taking an unintended action, misusing a tool, or operating outside its intended envelope. The unit a buyer ultimately trusts is therefore not a benchmark score but whether the agent decides the same way today and tomorrow, after a model update, after a vendor change, and after a team hand-off.

We call this property **coherence**: the same scenario producing the same governed decision regardless of which model or substrate produced the underlying text. Coherence is not accuracy, and it is not bit-identical output — language models do not reliably emit identical text even at temperature zero. It is *determinism at the behavioral level*: the action chosen, the policy applied, the route taken, and the escalation triggered are invariant under substrate change, even as surface wording varies. The engineering thesis of RCS reduces to a single sentence: **the model may phrase differently; the application decides consistently.**

This separation of concerns — a probabilistic executing layer supervised by a deterministic governing layer — is not novel in systems engineering; it is the standard answer wherever a capable-but-self-unaware component must be made dependable. Endpoints transmit while TCP congestion control governs flow so the network stays coherent; aircraft fly while air-traffic control governs spacing; equipment draws current while circuit breakers govern thresholds; services execute while a consensus layer governs ordering. In each case the executing layer is necessary but cannot govern itself, and a structurally distinct layer is required. Agents are the executing layer of the agentic economy; **RCS is the missing coherence layer.**

Before a system's behavior can be governed, a prior question has to be answered: whose behavior, and on whose behalf? Governance presupposes a represented entity — an actor with state to preserve and authority to enforce. A person, a company, a brand, an institution, or a device may each act in the world, and each carries identity, reputation, and continuity into every interaction. Yet digital actors today have no persistent presence of their own: their identity, memory, and behavior are fragmented across the applications they pass through and owned by each platform rather than by the actor. Identity is abundant; persistent, portable presence is scarce.

Rainfall addresses this through two complementary layers. *Aura* establishes a persistent, portable presence for an actor across applications, models, and platforms — the state and identity that governance acts upon. *RCS* governs how that presence behaves when it acts. The two are separable concerns that compose into a single deployable artifact: the Coherent Application (CApp), a governed agentic application whose behavior remains dependable despite changes in the underlying model or substrate. This paper concerns the second layer — the runtime — and develops it in technical depth; the first is named here because governance is only meaningful with respect to the entity it represents.

Section 2 explains why this layer had to become a self-contained engine. Section 3 gives the system overview. Section 4 details the coherence mechanism, Section 5 the per-step reflex lifecycle, Section 6 substrate-neutral orchestration, Section 7 knowledge and economics, and Section 8 decentralized deployment. Section 9 defines coherence formally and outlines evaluation. Sections 10–11 cover related work and limitations.

2. Design Rationale: From Add-on to Engine

RCS was initially conceived as an additive layer — a wrapper that would sit on top of existing agent platforms and orchestration frameworks and add governance. That design did not survive contact with

the requirement. Effective behavioral control has to act *before* an emission reaches the world, has to persist behavioral state across a model swap that happens mid-execution, and has to anchor audit at the granularity of each step rather than each session. These invariants cannot be guaranteed by a layer that merely observes another vendor's runtime after the fact. The depth at which governance must operate therefore mandated that RCS become its own engine — and, because it must coordinate work that may itself run on other orchestrators, **a primary orchestrator of orchestrators.**

A composition of separate vendors — one framework for control flow, one engine for durability, one tool for observability, one product for policy — carries as many contracts, release cadences, and security postures as it has parts, and the seams between them leak exactly the properties RCS exists to sell: cross-vendor latency, version drift, configuration divergence, and audit gaps where one tool's traces do not reconcile with another's policy decisions. Pre-emission verification performed at instrumentation latency is no longer pre-emission; it is post-hoc logging. Single ownership of the trust contract — one layer answerable for what an agent did and did not do — is achievable only in a self-contained runtime. The cost of that choice is real (more to build, a single throat to choke, broader architectural surface), and is acknowledged in Section 11.

3. System Overview

RCS positions itself as the governance, or *harness*, layer of the agentic stack (Figure 1). Beneath it sits a data layer (knowledge, tools, systems of record) and an intelligence layer of probabilistic models from one or more vendors. Base-model access is increasingly a commodity; the durable value is the layer that makes any model's behavior dependable. The deliverable that layer produces is the **Coherent Application (CApp)** — a governed agentic application that packages capability (what it does), behavioral continuity (how it stays itself over time), and the actor's sovereign presence (who it represents and how it shows up) into one deployable, portable artifact. The CApp is the unit of deployment, portability, and composition: it is what moves across substrates (Section 8) and what later composes with other CApps into larger governed systems (Section 12), with RCS the layer that keeps its behavior dependable as it does so.

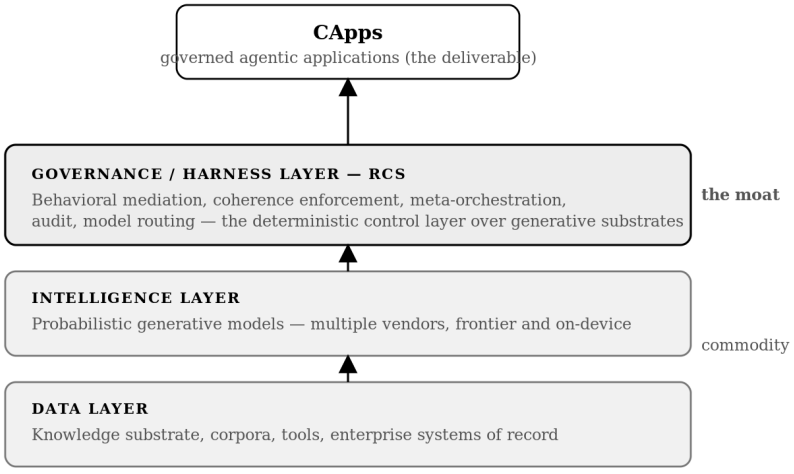

Figure 1. The agentic stack. RCS is the governance/harness layer that converts a probabilistic intelligence layer into dependable, deployable applications (CApps).

RCS is built on six platform principles, summarised by the acronym **COASTS** — Composable, Open, Authored, Social, Trusted, Specialised — and is realised as two cleanly separated tiers connected over a streaming RPC contract: an **Engine** that runs CApps and implements the principles, and a **Web Client** through which humans author and govern them. Authoring spans the full spectrum of users: a

structured editor for technical builders (point-and-click composition and skill-level scripting) and an intent compiler for non-technical authors, who describe a specification in natural language and have it compiled into a CApp. Table 1 maps the principles to their role.

Principle	Role in producing coherence
Composable	CApps are built from pluggable primitives (actors, choreographies, archetypes, adapters) that recombine without rewrite, so behavior is preserved across substrate change.
Open	The runtime executes above any vendor backend through bidirectional adapters; CApps move between substrates without rewrite.
Authored	Every CApp is authored, versioned, owned, and ejectable; authoring rights and lineage are first-class.
Social	Multi-organization coordination, fork lineage, and merge/split handling live in the substrate, not in a settings page.
Trusted	Identity, pre-emission verification, tamper-evident lineage, and cross-modal consistency are enforced structurally.
Specialized	Domain modulation profiles tailor behavioral inference to the operational domain (e.g., support, legal, finance).

Table 1. The COASTS principles.

4. The Coherence Mechanism

4.1. Surface-level modulation

RCS governs an agent without modifying its internals. A well-formed agent exposes a small number of *surfaces*: its inbound context, its specification (prompts/skills), its model boundary, and its tool-call boundary — everything that enters or leaves the agent. RCS wraps these surfaces and injects supervision at each one (Figure 2). The governing rule is conservative: **operate only at surfaces that can be observed and modulated, and never change an agent's intrinsic behavior directly.** Where a surface can be observed and mutated, RCS acts on it directly; where it cannot, RCS falls back to *prescriptive modulation* — it informs the agent, through its specification or prompt surface, of the required change, then monitors subsequent emissions to confirm the behavior actually shifted.

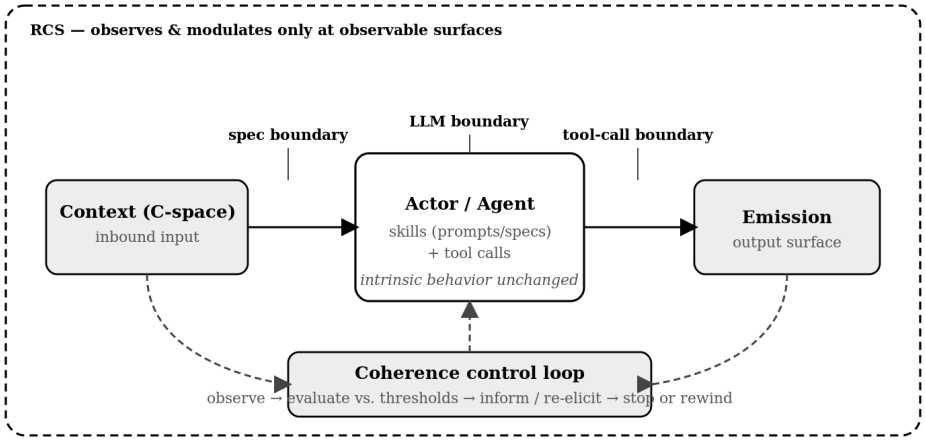

Figure 2. Surface-level modulation. RCS wraps the agent at the context, specification, model, and tool-call boundaries and runs an observe–evaluate–correct loop, leaving intrinsic agent behavior untouched.

The control loop is therefore: observe an emission at a surface; evaluate it against the active thresholds; if it conforms, pass; if it does not, inform the agent and re-elicit, or — where an action has begun or completed and a rule is violated — stop or rewind it. RCS does not silently overwrite an agent's output; it asks the agent to bring itself back within bounds, and escalates to hard controls (stop/rewind) only when supervision at the surface is insufficient.

4.2. The behavioral–context manifold and imprints

To evaluate "conformance" quantitatively, RCS represents the state of an interaction as a point in a combined vector space. One component captures *behavior* — how the agent is acting — and the other captures the *context* of the interaction, which is the inbound side the runtime watches. We denote the behavioral space B and the context space C; the runtime situates each emission in their combination.

$$Z = B \oplus C, \qquad B \in \mathbb{R}^n,\ C \in \mathbb{R}^m,\ Z \in \mathbb{R}^{n+m} \tag{1}$$

The specific dimensionality of these spaces is an implementation detail; what matters are the **imprints** — bounded regions of this space that encode an intended behavior together with the rules that define how far an emission may stray from it. Imprints, not the raw axes, are the substantive object; they are the learned, governed envelopes against which every emission is measured (Figure 3).

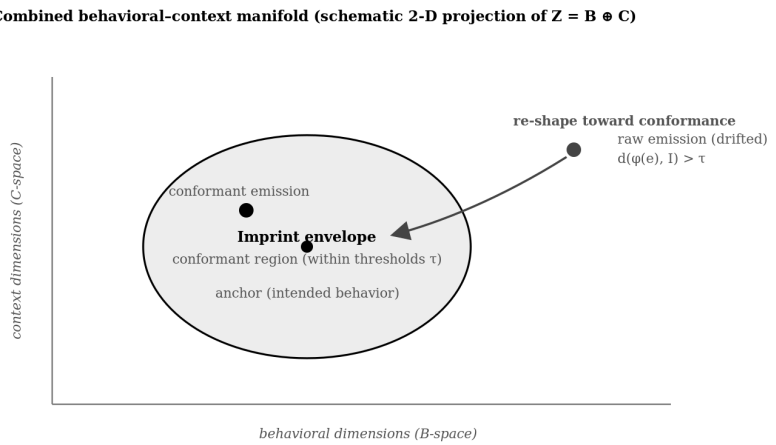

Figure 3. Schematic projection of the combined manifold. An imprint defines a conformant envelope around an intended behavior; a drifted emission is re-shaped toward conformance rather than overwritten.

Conceptually, an emission e is admitted when, for every active imprint, the distance between the emission's position $\varphi(e)$ and the imprint envelope stays within that imprint's threshold:

$$\text{admit}(e) \quad \text{iff} \quad d(\varphi(e),\, I_k) \le \tau_k \quad \text{for all } k \tag{2}$$

When an emission strays outside its envelope, the magnitude of the excess defines a drift severity, which selects the corrective action — re-elicit at low severity, escalating to stop or rewind as severity grows:

$$\delta(e) = \max_k \left[\, d(\varphi(e),\, I_k) - \tau_k \,\right]_+ \qquad \delta = 0\text{: pass} \quad \delta > 0\text{: re elicit / stop / rewind} \tag{3}$$

Equations (1)–(3) are a deliberately abstract formalization of the described mechanism, included to make the control logic precise; they are not the proprietary internals. The essential idea is that an imprint exerts a restoring force: it "shapes" an emission back toward the intended behavior whenever the emission drifts past threshold, which is why coherence is produced by construction rather than checked after the fact.

5. The Reflex Lifecycle

Surface modulation is applied uniformly through a fixed lifecycle that runs on every step an agent executes. RCS advances each step through an eight-stage **reflex chain**, preceded by a pre-flight battery of checks. Each reflex owns exactly one concern, and the order is fixed: behavioral state is consulted before a decision; inference runs once; policy is applied before emission (because catching a violation after emission is logging, not governance) and again after; behavioral state is updated atomically with the step; cost is metered per step; and the step is anchored to a tamper-evident ledger. Figure 4 shows the chain.

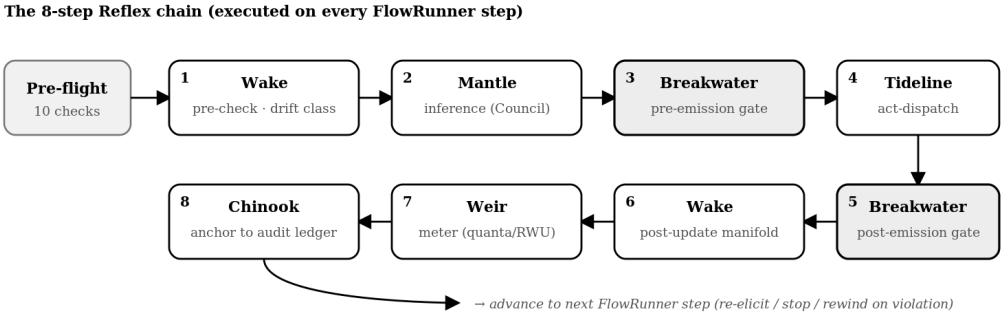

Figure 4. The eight-step reflex chain executed per step: state pre-check, inference, pre-emission gate, dispatch, post-emission gate, state update, metering, and audit anchoring.

Because every step emits the same eight-reflex receipt, the audit story is uniform across applications, substrates, and vendors: a reviewer asking "what happened on this turn?" receives the same structured answer every time. Two reflexes are policy gates (pre- and post-emission); on a violation the chain does not merely record it but acts on it — re-eliciting, stopping, or rewinding — so that a rule breach changes the outcome rather than just the log. This places a deterministic, non-bypassable governor on top of an otherwise probabilistic execution.

6. Substrate-Neutral Meta-Orchestration

RCS is a full orchestrator in its own right, incorporating the primitives offered across existing engines — on the order of sixty distinct orchestration primitives spanning business-process and workflow patterns drawn from durable-execution engines and the major cloud and enterprise platforms. It was driven to build its own orchestrator for two reasons that surfaced when delegating to third parties: they do not share a common construct language (each substrate demands different constructs, forcing a proliferation of adapters, fallbacks, and buffers), and they are not uniformly deterministic. RCS therefore offers a single, consistent construct set and runs workflows natively when that is fastest.

Crucially, native execution is a choice, not a lock-in. A choreography node may *hand off* to an external engine, which RCS treats as a *provider* encapsulated inside that node. The node becomes opaque: RCS does not see inside the external execution and resumes only on the hooks or notifications the provider returns. If a provider returns nothing, the node simply remains opaque. The same CApp can thus run entirely inside RCS or delegate any node to an external substrate, and a single agent definition can

span vendor backends — model providers, cloud agent platforms, frameworks, and workflow engines — through one dispatcher (Figure 5).

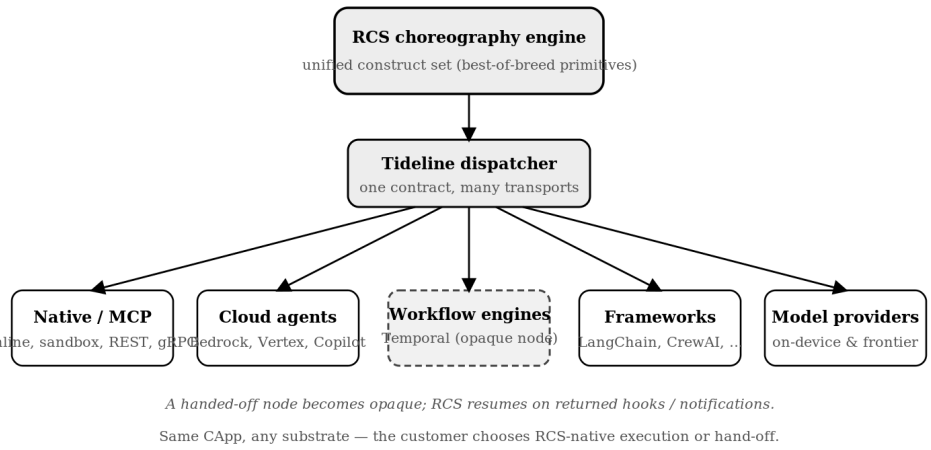

Figure 5. Substrate neutrality. One dispatcher binds a CApp to many transports; a node handed off to an external engine becomes opaque and resumes on returned hooks.

7. Knowledge, Model Routing, and Economics

Two further subsystems make coherent execution practical at scale. The first is a **knowledge substrate**: a governed store assembled by a pipeline that normalises source material (resolving and correcting named entities through web and model checks), extracts every named entity and event, dispatches a research agent to summarise supporting context, and writes the result to a single source of truth using an open, file-based knowledge-graph format. This gives a CApp a verifiable, inspectable basis for what it knows, with clear provenance for where each fact originated.

The second is a model-routing and accounting subsystem. An **LLM council** selects, per request, an appropriate inference engine for the constraints the author has set, optimising for cost subject to quality and policy requirements, while the runtime tracks the *quanta* consumed by each action. This both lowers operating cost and provides the metering substrate for usage-based billing rather than flat subscription. Conceptually:

$$m^{*} \;=\; \arg\min_{m \,\in\, \mathcal{M}} \mathrm{cost}(m) \quad \text{s.t.} \quad c(m)\ \text{holds}, \qquad q \;=\; \sum_{a} \mathrm{quanta}(a) \tag{4}$$

Because routing is constraint-driven, the same CApp can prefer a low-cost or on-device model for routine work and reserve frontier models for requests that demand them — a property that becomes central to deployment (Section 8).

8. Decentralized, Edge-Capable Deployment

RCS is decentralized by architecture and can run anywhere. Its heaviest dependency is an embedded, verifiable document store; by adopting a lighter native (Rust) build of that store, the engine requires no hardware accelerators. As a result the runtime executes not only on cloud servers but on desktops and edge devices, and on phones as a system library rather than an application. Combined with constraint-driven routing, this lets a deployment resolve inference locally — using an on-device model where available — and reach out to frontier models only when a request requires it (Figure 6).

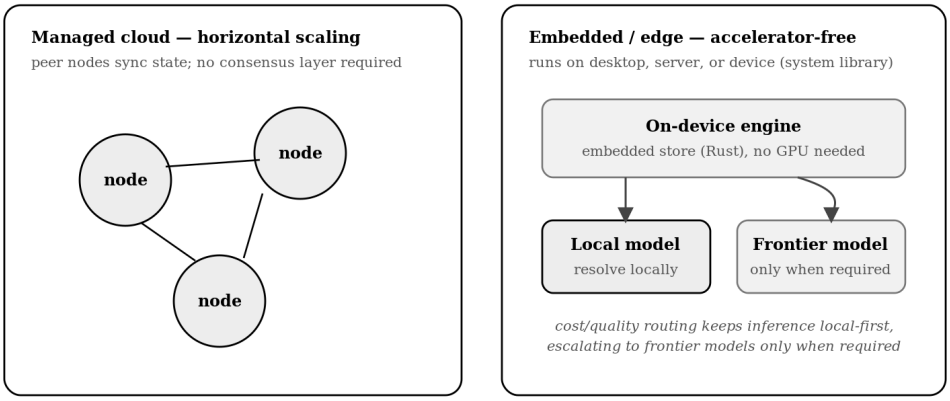

Figure 6. Deployment topology. Cloud deployments scale horizontally with syncing peer nodes and no consensus layer; embedded deployments run accelerator-free on desktop or device with local-first inference routing.

For availability, cloud deployments scale horizontally: several peer nodes run and synchronise state with one another, and because the design does not require global agreement, there is no consensus layer to coordinate. The same engine subsystems run in every mode; only the deployment substrate differs, which is what allows a CApp to move from managed cloud to an embedded, sovereign, or air-gapped deployment without being rewritten. The architecture is more involved than a purely centralised service would be, and that complexity is a deliberate investment in the edge and sovereignty future it enables.

9. Coherence: Definition and Evaluation

Example: A decision-coherence failure. An automotive service agent was presented with customer financial information that was contextually valid but operationally irrelevant to vehicle valuation. The agent incorporated that information into a purchase recommendation, producing an action inconsistent with the governing business policy. The failure was not one of model capability: the underlying generation was fluent and plausible. It was decision non-invariance — the system selected an action outside the authorized decision boundary, of a kind that may resolve differently across runs, prompts, or model versions. In the terms developed below, D(s; σ) was not invariant where policy required it to be. This is the precise failure RCS is built to eliminate: not to make the text correct, but to make the governed decision the same every time.[1]

1. Illustrative scenario, representative of a documented class of agentic failure — an autonomous service agent conflating a financing balance with a valuation figure. It is presented to characterize the failure mode, not as a specific case study.

We can now state coherence precisely. Let Σ be a set of substrates (model versions, vendors, deployment modes) and let D(s; σ) be the governed decision an application makes for scenario s on substrate σ, with T(s; σ) the surface text it emits. An application is coherent over Σ when the decision is invariant across substrates even though the text need not be:

$$\forall \sigma_i, \sigma_j \in \Sigma : \quad D(s;\sigma_i) = D(s;\sigma_j) \quad \text{while} \quad T(s;\sigma_i) \neq T(s;\sigma_j) \tag{5}$$

This yields a concrete evaluation methodology. Coherence can be toggled on and off within a CApp, which makes the contribution of the governance layer directly observable: with governance disabled, raw emissions that transgress a boundary appear; with it enabled, those emissions are gated, re-elicited, or rewound. An instrumentation panel surfaces, for any output, which guardrails were invoked in producing it and what raw output transgressed a boundary — turning "coherence" from a claim into a visible, before/after comparison.

At the portfolio level, the same evaluation generalises to a cross-substrate scenario pack: run a fixed battery of scenarios for one CApp across multiple model and vendor backends and measure per-decision divergence (target: zero on action selection, policy application, escalation, and refusal) while allowing surface text to vary within fidelity bounds. Each step's audit receipt provides the evidence; a coherent application produces the same governed decisions and the same receipts across substrates.

10. Related Work and Positioning

RCS shares a surface with several adjacent categories but is identical to none of them. Table 2 summarises the distinction; the defensible position is the *composition* — surface-level coherence enforcement, a behavioral manifold, a non-bypassable reflex chain, substrate neutrality, and structural audit operating together — rather than any single capability.

Category	Examples	How RCS differs
Orchestration frameworks	LangChain/LangGraph, CrewAI, AutoGen	Frameworks are developer kits that produce code and offer basic, non-deterministic control flow; RCS is an opinionated runtime with a non-bypassable governing layer and a consistent construct set.
Agent platforms	Cloud and CRM agent platforms	These are substrate-bound by design; RCS is substrate-neutral and can run one CApp across competing backends, treating each as a provider.
Durable-execution engines	Temporal, Restate, Inngest	These make any code durable but do not mediate output or hold behavioral state; RCS treats them as providers it can hand off to inside a choreography node.
Governance / observability tools	Tracing and policy products	These sit beside the agent and are post-hoc by architecture; RCS embeds verification before emission, as part of the runtime.

Table 2. RCS versus adjacent categories.

11. Limitations and Forward Work

- **Self-containment cost.** Owning the full runtime is heavier to build and operate than a thin wrapper, and concentrates accountability in a single layer. The bet is that the seam-leakage of composed alternatives makes that cost worthwhile; if it does not, a lighter wrapper wins.
- **Imprint maturity.** The behavioral manifold and its imprints are tuned empirically and continue to evolve as real-world data accrues; their specifics are expected to refine with further testing.
- **Surface-only control.** Because RCS never alters an agent's internals, control is bounded by what is observable and modulable at the surface. Where direct mutation is impossible it relies on prescriptive modulation and monitoring, which is best-effort rather than guaranteed.
- **Platform constraints at the edge.** System-library deployment on some mobile platforms depends on platform permissions outside the runtime's control.

© 2026 Rainfall Inc

- **Evaluation breadth.** The cross-substrate scenario-pack methodology is sound in principle; broad, independent, production-scale measurement is the natural next step.

12. Conclusion

Generative intelligence is mercurial; production systems are not allowed to be. RCS resolves that tension by making coherence the responsibility of a separate, deterministic control layer that supervises probabilistic agents at their surfaces, situates their behavior in a governed manifold, enforces conformance through a fixed per-step reflex chain, orchestrates work across any substrate, and runs from cloud to edge. The result is a runtime that brings the trustworthiness of a deterministic system to the power of generative AI, and a deliverable — the CApp — that carries that trustworthiness with it wherever it runs. The contribution is less any single primitive than the demonstration that coherence can be produced *by construction*, as a structural property of the runtime, rather than hoped for prompt by prompt.

The implications extend well beyond any single deployment. Once behavior can be governed as reliably as computation, whole classes of work that have so far resisted automation — regulated, high-stakes, multi-party, and long-running — become addressable, because the question shifts from "can the model do it?" to "can we trust the system to do it the same way every time, and prove that it did?" A dependable behavioral layer lets organizations delegate real authority to agents rather than supervising them turn by turn, and lets that delegation cross the boundaries — between vendors, teams, and institutions — where today's single-vendor, single-machine agents break down.

As coherent applications accumulate, they begin to compose. A CApp authored once can be extended, federated, and combined with others into larger coherent systems whose collective behavior remains governed, portable, and auditable — the multi-agent, multi-organization fabric an autonomous economy will require. In that sense RCS is not only a runtime for individual agents but a foundation for building coherent agentic systems at large: an infrastructure layer on which trustworthy AI behavior can be authored, owned, and carried forward — a step toward an agentic future that is dependable by construction and sovereign by design, rather than concentrated in a handful of opaque providers. That is the outcome Rainfall is building toward.

It is worth closing on the convergence the paper has been building toward. Section 1 established that an actor's presence must persist across applications and substrates; Section 9 showed formally that an actor's decisions can be made to persist across them as well. These are the same property — persistence — applied to two objects: who is acting, and how they decide. A Coherent Application is the artifact in which both hold at once: the same actor, deciding the same way, wherever it runs. Trust is what accrues when presence and decision persist together, and that is finally what Rainfall is building — not a model, and not a wrapper, but the layer where presence and decision are held coherent over time.

© 2026 Rainfall Inc

Appendix A. Glossary

Term	Meaning
Coherence	Same scenario → same governed decision across substrates, while surface text may vary.
CApp	Coherent Application: a governed agentic application packaging capability, behavioral continuity, and sovereign presence as one portable artifact.
Aura	A persistent, portable presence for an actor (person, organization, brand, or device) across applications, models, and platforms — the state and identity that governance acts upon.
Sovereign presence	Who a CApp represents and how it shows up; the actor's owned, portable identity established by Aura — distinct from capability (what it does) and behavioral continuity (how its behavior persists).
COASTS	The six platform principles: Composable, Open, Authored, Social, Trusted, Specialized.
Imprint	A bounded region of the behavioral–context manifold encoding an intended behavior and its conformance thresholds.
B-space / C-space	Behavioral and context vector spaces; emissions are situated in their combination.
Reflex chain	The fixed eight-step per-step lifecycle (state, inference, pre-gate, dispatch, post-gate, update, meter, anchor).
Prescriptive modulation	Correcting behavior indirectly by instructing the agent through its spec/prompt surface and monitoring the result.
Provider hand-off	Delegating a choreography node to an external engine, which becomes opaque and resumes on returned hooks.
Quanta	The metered units of work consumed per action, used for cost optimization and usage-based billing.

Appendix B. Notation

- $\mathbf{Z = B \oplus C}$ combined state of an interaction; $B \in \mathbb{R}^n$ (behavior), $C \in \mathbb{R}^m$ (context).
- $\boldsymbol{\varphi(e)}$ the position of emission e in the combined manifold.
- $\mathbf{I_k}$, $\boldsymbol{\tau_k}$ the k-th active imprint and its conformance threshold.
- $\mathbf{d(\cdot,\cdot)}$, $\boldsymbol{\delta(e)}$ distance to an imprint envelope, and the aggregate drift severity of an emission.
- $\mathbf{D(s; \sigma)}$, $\mathbf{T(s; \sigma)}$ the governed decision and the surface text for scenario s on substrate σ.

This white paper synthesizes the Rainfall Coherence System engineering canon with primary engineering commentary. Equations (1)–(5) are representative formalizations of the described mechanisms, provided for precision; they are not implementation disclosures. *© 2026 Rainfall Inc.*