Composer: Architecture and Operation of an Agent-Oriented Workflow and Coordination System

Working technical report - public attachment edition, 2026-10-30. The supplied manuscript dates to 2026-10-28; this edition incorporates an October 30 receipt review and original-author clarification. Historical results retain their dates and scope. "Certified" means internal project review: the builder built, tested and self-reviewed, then an independent auditor who did not build it reviewed the work. It does not mean scholarly peer review or endorsement by Meta, Cerebral Valley or any organizer. Evidence labels describe the cited project records, not a new replication of every result for this edition. Private references and abbreviated hashes describe sources; they are not public reproduction links or complete artifact identities. "Factory" remains a working metaphor. Public omissions are marked.

S. Luc Misfit Circuit, Columbia, SC, USA *stephen@misfitcircuit.com*

Manuscript prepared with the Composer crew. Evidence classes are labeled throughout in the author's house style: PROVEN (reproduced by an independent checker), REPORTED (crew-attested, not independently reproduced), and UNMEASURED (believed, not yet instrumented). The author regards unlabeled claims in systems papers as a defect and declines to commit it.

October 30 update: installed scope and historical limits

Receipt review records installed native-worker jobs in the real UI, output on the Linux W530 computer, coordinator/readiness recovery, and installed text-only provider plug-in jobs, including Muse Spark and a configuration-only local stub. A built-in SWE job also survived the plug-in install. Compatible OpenAI-style chat-completion APIs can be added by configuration; other protocols require adapters. These plug-ins return text, without filesystem, browsing or an executed tool loop. An installed OpenAI API job used a PAY_PER_USE route; the old subscriptions-only account is no longer current. Model identity is provider-reported, not independently established from model internals.

Capacity is a dated observation, not a universal ceiling: the campaign finalized October 27 UTC verified 16 overlapping Linux agents. Subsequently installed operating settings were Mac 4 slots and Linux 9, with holds and shared account/model pools further limiting availability. The reported 195 projects are the October 28 source count, not a fresh inventory. The save/review/Start write surface is explicitly granted and supports a simple workflow slice; its complete start/replay/output proof used a copied engine and fake CLI, not a fresh live-provider execution. The historical sections below retain the build-stage context where identified.

Sources: internal installed plug-in final results and independent closure (October 28); installed OpenAI route results/closure (October 28); Linux acceptance and cross-host final report (October 26-27); installed workflow proof (October 28), reviewed October 30. No new provider requests were made for this edition.

Abstract

We present Composer, a self-hosted system for commissioning, executing, and verifying software work performed by crews of heterogeneous large-language-model agents under a single human operator. Composer's design begins from an empirical complaint: in naive multi-agent setups, the agents spend a substantial fraction of their capacity *orchestrating each other* - relaying, babysitting, and re-establishing context - rather than producing work. Composer attacks this overhead structurally. Work is commissioned through a dashboard into projects; execution is divided between *rooms*, in which resident agents conduct long-running sessions, and *queue seats*, which execute bounded dispatched tasks under capacity limits and leases. Providers are abstracted by a strict doctrine - **the model is the seat; routes are doors** - so that any model may be reached through any transport that carries it, and a recently added plug-in socket admits new providers behind a uniform receipt contract without engine modification. A save → review → explicit-start workflow contract, keyed to revision/spec/manifest content identity, lets a human interface and an agent interface drive the *same* run and prove that they did. Governance is receipt-based: builders build, test, and self-review; independent auditors - frequently blind reviewers drawn from a *different model family* - certify. We report the system's operating envelope (two computers under one central hub; 195 projects reported in the October 28 source; a later reviewed capacity record verifying 16 overlapping Linux agents), a documented adversarial review run (7 blind reviewers, 2 model families, 18 findings, 16 fixed, 2 deliberately deferred), and the honest limitations of the current build, including a structurally hard-wired legacy queue contract and the residual - reduced, not eliminated - orchestration tax.

Index Terms - multi-agent systems, LLM agents, agent workflows, provider abstraction, verification, receipts, human-on-the-loop.

1. Introduction

The premise of agent crews is division of labor. The practice, as observed by the author over roughly five months of operating a named crew of agents (Codex-family and Claude-family seats working the same problems), is that division of labor silently re-imports the coordination costs it was meant to abolish. Agents relay messages between agents. Agents wait on agents. Agents re-derive context that a sibling established an hour earlier. The author's summary of the failure mode is the founding requirement of the system described here:

> "A lot of agents babysit it when it really could be left alone."

Composer began, accordingly, as tooling so the crew would do *less orchestrating and have better tools*. It has since become an agent-oriented workflow and coordination system with a stated north star against which every feature is judged:

1. **It runs itself.** Steady-state operation requires no human in the loop.
2. **Humans receive exceptions only.** The operator's attention is a scarce resource, spent on decisions, not on transport.
3. **It is operable by the operator unaided.** A factory its owner cannot run is a dependency, not an asset.

These are not aspirations appended to a finished system; they function as an acceptance test. A mechanism that increases operator load is, by definition, a regression - a criterion we apply in §7 to the factory's own management tooling, with an unflattering result the author has chosen to publish anyway.

Contributions. (i) A dual execution model - resident *rooms* and dispatched *queue seats* - sharing one project and identity substrate (§3). (ii) A provider doctrine and plug-in socket in which every call, whatever

the transport, emits a uniform receipt recording requested versus *response-reported* model identity, with mismatches flagged rather than smoothed over (§4). (iii) A workflow lifecycle contract - save, review, explicit start - in which two different client surfaces are proven to drive one and the same run, reconciled by content hash (§5). (iv) A governance layer in which certification is structurally separated from construction and preferentially assigned across model families, under a small set of operator laws (§6). (v) An evaluation reporting the operating envelope and a fully documented adversarial review run, plus limitations stated at the same prominence as the results (§7, §8).

2. Background and Related Work

Composer sits at the intersection of three lines of work. First, *agent frameworks* (orchestration libraries and autonomous-agent loops) typically optimize the single-agent or single-pipeline case and treat cross-agent verification as the user's problem; Composer inverts this, treating verification as a first-class subsystem and raw generation as a commodity supplied by seats. Second, *CI/CD and build farms* contribute the queue, lease, and artifact-identity machinery, though classical build farms execute deterministic jobs, whereas Composer's workers are stochastic and occasionally inventive in ways that must be caught (§6). Third, the author's own adjacent systems - a facts-only session *excavator* that grades historical agent verdicts from sealed records, and a deterministic *scoring layer* for claim support - supply the evidentiary culture (hashes, sealed chains, labeled proof states) that Composer's receipts inherit. We know of no deployed system that combines all three under a single non-engineer operator, though we do not claim the negative strongly (UNMEASURED: no systematic survey was undertaken; the author's literature review consists largely of having lived the alternatives).

3. System Architecture

3.1 Substrate and topology

Composer is deployed across two computers - the operator's Mac (primary) and the Linux W530 computer - under **one central hub**, holding **195 projects** in the October 28 source (REPORTED: operator's console count, not recounted for this edition). Projects are the unit of commissioning: a project carries its specification, its revision history, and its runs. A dashboard provides the operator's surface; an MCP endpoint provides read-only discovery, description, and reads to authorized external agents, deliberately without write authority (the write path is a separate, separately granted contract; §5).

3.2 Rooms

A *room* is a persistent session space in which a resident agent works a problem over time - the digital analogue of an apartment, in the codebase's own vocabulary (roomapt.py), complete with per-room services for queuing work to the room's seat. Rooms suit open-ended, context-heavy work: the resident accumulates the local knowledge that a fresh dispatch would have to re-derive. The room's corresponding risk - context drift and the quiet inheritance of its own earlier decisions - is mitigated procedurally rather than architecturally: audits of room work are always conducted from **fresh sessions** (§6), a rule the operator learned empirically after observing that a continuing session "inherits the same decisions it typically makes."

3.3 Queue seats, dispatch, and capacity

A *queue seat* is a bounded worker: a task is dispatched, executed, and receipted, and the seat returns to the pool. The dispatch layer (queue_dispatch, a seat runner service, and a capacity module) enforces a concurrency ceiling and manages *leases* - a dispatched task holds its seat under lease, and held or stale leases are treated as exceptional state requiring operator authority to touch, never silently reaped by tooling. The capacity campaign finalized October 27 UTC verified **16 overlapping Linux agents** within its tested hardware/route scope. This is an observed overlap, not a universal ceiling. Later installed operating settings were **Mac 4 slots and Linux 9**; unresolved holds and shared account/model pools can reduce available capacity (§7). The earlier manuscript reported approximately 30 seats exercised across the fleet under rotation.

3.4 The sandbox discipline

No component of Composer is edited in place. Development proceeds in a **sandbox**: a full copy of the relevant source (in the most recent major build, 906 files spanning the app, the dashboard UI, the connector, and the engine), edited and tested there, packaged with a manifest, and installed only after independent checks pass. The live system is never the working tree. This discipline costs disk and patience; it has repeatedly converted what would have been production incidents into diffs.

4. Providers: Seats, Routes, and the Plug-In Socket

4.1 The doctrine

Definition 1 (Seat). A *seat* is a model in a role: the reasoning entity whose judgment a task requires.

Definition 2 (Route). A *route* is any transport that reaches a seat: a vendor API, a CLI subscription, an aggregator plan, or a local plug-in.

Composer's foundational provider claim is that seats and routes are independent axes. If a model is unreachable via one route, it should be reachable via another carrying it; route substitution across subscription transports to reach the *same model* is intended resilience, not an anomaly. A corollary the operator states as a correction to a common misreading: the mechanism behind a door (an API key, a CLI) is not the billing behind the door, which in the earlier manuscript was described as subscription-based. The October 28 manuscript described a subscriptions-only policy. That statement is superseded: the installed OpenAI API route is explicitly **PAY_PER_USE**. Billing depends on the route; neither an API transport nor an earlier credit allowance establishes free current use. List-rate estimates are distinct from verified billing. Remaining credits and new-grant activation were not checked in this edition. Seat inventory at the time of writing includes seats reached via a coding-assistant CLI (Devin), an aggregator plan (OpenCode Go), a coding-plan API (Z.AI, carrying the GLM family), a prepaid API (DeepSeek), and - via the socket below - the Meta Model API (Muse Spark and Muse Wonder).

4.2 Legacy adapters and their discontents

Composer's built-in execution surfaces (rooms and queue seats) historically wired providers directly, with adapter kinds per transport family. The wiring accreted two structural defects, documented here because the socket's design is unintelligible without them. First, a particular legacy adapter's queue contract is *hard-wired to a single model* (the queue's model constant names one GLM variant), so "the queue accepts a provider" was, for any other provider, false by construction. Second, the queue path offers *no per-provider launcher seam*: there is no joint at which a new provider can be attached to dispatch without editing dispatch. These were adapter-specific findings in the historical integrating-engineer review. They do not describe the entire installed queue: the newer installed dispatch has plug-in work and readiness branches, with real UI jobs independently checked.

4.3 The plug-in socket

The socket (plugin_socket.py) admits a provider as a **plug-in**: a folder containing a manifest (plugin.json) and one runner. The engine integrates the socket exactly once; each plug-in then integrates zero times. Integration into the queue dispatch is additive and minimal - in the reference integration, the sole modified engine file carries a **19-line diff against pristine, all insertions**, built-in branches untouched, and rollback consists of disabling the socket. Every candidate route ships enabled: false; nothing about the socket authorizes or performs a live change.

A plug-in call emits a uniform **receipt** (RECEIPT.json) whose fields encode the system's evidentiary rules:

- **Requested versus resolved model.** The receipt records both the model the caller requested and the model the *response reports* having served the call. A mismatch is flagged, never normalized away. (The receipt does not trust the request line to describe reality - a small rule with large consequences for audit.)
- **Reason vocabulary.** Outcomes are expressed in the queue's own reason_code vocabulary, so plug-in results compose with built-in results in downstream tooling.
- **Cost.** Cost is recorded at the provider's recorded rate, or as *unknown*. It is never recorded as zero by default: a fabricated zero is a claim, and a false one.
- **Credentials.** The receipt and all logs name environment variables only; a redactor processes every written string. In the reference plug-in's offline test suite, a test in which the provider echoes an Authorization header back is included specifically to prove the redactor catches it (PROVEN: suite 5/5, including that test).
- **No internal retries.** A plug-in that fails reports failure. Retry policy lives one layer up, where it can be seen.

Two plug-ins ship in the reference package: an *echo* stub (the control group of provider plug-ins) and the *Meta* plug-in, exposing Spark by default with Wonder selectable; tool definitions are forwarded to the model but tool calls are recorded to a side file and **never executed** - a plug-in seat in v1 is a reasoning seat, not an acting one. Deliberate omissions in v1 (streaming, tool execution, provider discovery, credential storage) are enumerated in the plug-in specification as *omissions*, on the principle that an unlisted absence is a bug and a listed one is a design.

Live evidence. Through the integrated runner (not merely the standalone plug-in), both Meta seats answered a smoke prompt with HTTP 200 and the exact expected reply string, at a combined notional cost of $0.002611 at recorded rates (REPORTED: integration build report, tested by the integrator; this passage alone is not an independent reproduction). That build-stage report described a fail-closed canary and room/direct-only access. **Superseded installed status:** final repair and independent closure record plug-in dispatch/readiness integration and real UI jobs for Muse Spark plus a configuration-only local stub. The

compatible API plug-ins remain text-only; they do not execute tool calls or provide filesystem/browsing access. Final plug-in tests were 28 plus UI 100; the broader comparison retained 12 baseline failures, with no additional failures in that comparison. This is not a clean full-suite pass or proof of no defects.

5. The Workflow Contract: Save, Review, Start

The factory's most recent major build adds a write path where previously only reads existed, and its contract is the paper's clearest example of the house style. For one existing class of simple single-worker workflows, three operations are defined:

- composer_save_draft - persist a draft of the workflow;
- composer_review_project - render the draft for inspection *as the engine itself understands it*;
- composer_start_project - explicitly start a run.

Three properties are load-bearing. **One implementation, two surfaces.** A single module implements the contract, consumed identically by the human dashboard (the "Steve UI") and by agent-facing action surfaces. There is no second start state machine and no direct writing of queue files by clients; starts ride the engine's own machinery. **Identity by content.** Drafts and runs are keyed to the engine's revision/spec/manifest identity, so "the run you reviewed" and "the run that started" are the same object in the only sense that matters - byte identity - rather than in the conversational sense. **Reconciliation.** Both surfaces reconcile against the same run record after start.

Proof (fake worker, labeled as such). Both client paths executed save → review → start and produced the *same run*, whose result hash matched across the two paths and against the artifact on disk (PROVEN within the sandbox; the worker was synthetic, and the report says so on its face). The edge battery accompanying the proof covers the unglamorous cases where such contracts usually die: revoked access between review and start (after the fix described below, **zero** dispatches occur), oversized requests (one bounded aggregate limit, 32 KB, enforced at a single point for both surfaces), and interrupted UI flows.

The independent check of this build - conducted Mac-side by two crew seats other than the builder - returned five findings, all fixed in a single delta batch, and the catalogue is instructive enough to summarize: a time-of-check/time-of-use gap in which the writer re-read an access file after validation (fixed by pinning the grant's fingerprint and revalidating synchronously at dispatch; a post-dispatch revocation is now honestly labeled *uncertain/effectual - no rollback claimed*, because that is what it is); a startup path that risked provisioning credentials implicitly (fixed: startup creates no file, token, or grant, ever; write access is a named opt-in via environment configuration); an inconsistency between surfaces in request-size handling (fixed by the single bounded limit above); and two defects that turned out to be *pre-existing in the baseline UI* rather than introduced by the build - a default project name derived unsanitized from model output, and a stale continuation in which a delayed navigation could act on a selection the operator had already left (fixed with a selection-generation guard: stale continuations now apply nothing, and the saved draft remains listed and recoverable). The candidate was then installed on the production Mac by the crew's integration owner after the independent checks, closing the loop: specification → sandbox build → independent audit → bounded repair → install, with the original engine process untouched throughout (REPORTED by the installing seat; install receipts on file).

6. Governance: Receipts, Laws, and Adversarial Review

Composer's governance assumes its workers are able and fallible, including about their own work. Its rules - the operator's laws, dictated and codified - are few enough to print:

- **Law 0 - Specs don't sit.** A finished specification is routed to a builder or carries a named block. An unrouted spec is a defect; a named block is a state.
- **Law 1 - Restore before patchwork.** A broken thing's first move is its backup. Patchwork is what remains after restore is ruled out.
- **Law 2 - Done has four parts.** The builder builds it, tests it, and self-reviews it; then an *independent* auditor certifies it. The builder never certifies their own work. Two corollaries: the builder works from the operator's **exact request, passed intact** through any relay; and testing happens **at the surface the operator uses** - a user-facing surface must be proven working *in that surface*, since code-level tests that assume the surface works are precisely where issues arise.
- **Fresh sessions for auditors.** Each audit begins in a brand-new session carrying only the job's files, because a continuing session inherits the decisions it is meant to examine.
- **Receipts or it didn't happen.** Claims carry their evidence class; deliverables carry full content hashes (an abbreviated hash is not an identity - a rule adopted after a real collision of confidence at eight characters); transfers are verified by read-back hash before being reported complete.

Cross-family review is the governance layer's distinctive mechanism. Because the crew spans two model families whose failure modes are imperfectly correlated, consequential work is audited by a seat from the *other* family, frequently **blind** - the reviewer receives the artifact, not the builder's narrative. A designated approver seat (the Sentinel pattern) reviews proposed changes in sensitive loops and approves the *exact* change before anything applies. Disagreement between families is treated as signal, not friction: the crew's creative geography literalizes the arrangement - two houses on opposite sides of a fictional hollow, one cafe in the middle where the families meet and ship together. The author is aware of how this sounds and notes, in the crew's defense, that the cafe has excellent throughput.

7. Evaluation

We evaluate three things: the operating envelope, one documented adversarial review run, and the regression discipline of the two most recent builds. We do not evaluate agent output quality in the large; the factory's position is that quality is established per-artifact by certification (§6), and an aggregate quality figure computed by the system about itself would violate its own evidence rules.

7.1 Operating envelope

Quantity	Value	Evidence
Computers under one hub	2	REPORTED
Projects held in October 28 source	195	REPORTED (console count; not recounted)
Observed Linux overlap	16 agents	Verified in campaign finalized 2026-10-27 UTC; not a universal ceiling
Later installed operating slots	Mac 4 / Linux 9	Receipt-reviewed; holds and shared pools constrain availability
Seats exercised under rotation	~30	REPORTED
Historical plug-in route smoke	Spark + Wonder, HTTP 200, exact reply	Builder-tested report; independent reproduction not established by this passage
Installed plug-in UI work	Spark + configuration-only local stub	Internal independent closure; text-only scope
Workflow contract	Both surfaces → same run, hash-matched	PROVEN (synthetic worker, labeled)

7.2 Case study: the September 26 run

The operator's public site documents a complete adversarial review run from the system's earlier operation: **7 blind reviewers drawn from 2 model families** examined a body of work and produced **18 findings**, of which **16 were fixed** and **2 deferred on purpose** - deferred being a decision with a reason, not a euphemism for lost. The run's check suites passed **86/86** and **49/49**. We cite this run because it is the factory's method in miniature: plurality of reviewers, independence of families, findings as countable objects, and deferral as a first-class outcome. (Source: the operator's own published account; figures as published there.)

7.3 Regression discipline

For the plug-in integration, the integrator ran the engine's full operator-services test discovery on an untouched baseline copy and on the candidate. Baseline: 998 tests with a pre-existing failing set of 68 (16 failures, 52 errors, 1 skip - including a known cluster of three approval-wiring failures, independently confirmed as exactly that cluster). Candidate: 1,014 tests with a failing set **byte-identical** to baseline - the integration introduced zero regressions while adding its 16 - and the socket's own module suite passed 17/17 with a synthetic end-to-end queue run completing with field-by-field receipt assertions (PROVEN: integration build report). For the workflow build, the candidate's builder suite matched pristine test-for-test (41/41 both), with the build's own new suites green (15/15, 10/10, and the UI suites 29/29 scoped) and the single full-UI failure shown to be *pre-existing on the untouched tree* (a packaging probe for a file the candidate does not ship). In both builds, the headline is not the pass count; it is that the *delta against baseline* is the reported figure. A green suite on a modified tree proves little. A zero-delta failing set against a measured baseline proves the thing a factory actually needs to know.

7.4 The babysitting metric, honestly

The founding complaint (§1) admits a metric - operator interventions per unit of shipped work - and the author regrets to report it is **UNMEASURED** as a continuous quantity. The indirect evidence is mixed and we present it as such: in the factory's favor, runs now execute end-to-end (commission → build → cross-family audit → bounded repair → install) with the operator acting only at decision points; against it, the operator's own agent (the author of the architecture notes, not of this paper's claims) recently required a dedicated *results tray and desk* - a derived board collecting lane results, checks, and open decisions - precisely because herding lane completions by hand had become a job in itself. The desk's existence is evidence that the orchestration tax was reduced and *relocated*, not abolished. Its derived board, we note with approval, fails closed: it reports zero provable closures rather than inferring them, because its linkage rule demands a recorded match. A factory that knows the difference between "done" and "provably done" is most of the way to the north star; the remaining distance is measured in exactly such admissions.

8. Limitations

1. **Provider scope remains bounded.** A particular legacy adapter has one-model wiring; this is not a restriction on the entire installed queue. Installed plug-in work/readiness integration and real UI jobs supersede the earlier room/direct-only limitation. Configuration-only additions support compatible OpenAI-style chat-completion APIs; other protocols need adapters. Plug-in outputs are text, without an executed tool loop, filesystem or browsing. These checks do not establish arbitrary workflow support.
2. **Stale route evidence.** Provider behavior is verified at build time; routes drift. One installation report in our records states, with a candor we endorse, that its full start-to-output proof used a copied engine and a fake CLI and makes *no fresh real-provider claim*. Consumers of factory output must read receipts for their evidence class, not merely their verdicts.
3. **Single operator.** Operation by one person unaided remains a stated goal, not a measured universal result; dependence on one operator is also the system's bus factor. The crew mitigates (documentation, receipts, sealed checkpoints), but the operator remains a single point of judgment.
4. **Fresh-session audits cost context.** Independence is bought by discarding the auditor's accumulated context on every run. We judge the trade correct and note it is a trade.
5. **The operator's laws are enforced socially and procedurally**, not mechanically, except where tooling has been built to carry them (receipts, hash rules, the desk's linkage). A determinedly careless operator could defeat them. The operator is, however, the author, and is disinclined.

9. Conclusion

Composer's thesis is that a software factory for agent crews is primarily an *evidence machine that also produces software*. Its rooms and seats produce work; its receipts, manifests, sealed cases, and cross-family audits produce the right to believe the work. The system's north star - runs itself, humans get exceptions only, operable by its operator unaided - remains a direction rather than a destination, and this paper has tried to report the distance remaining with the same receipts discipline the factory demands of its seats. The babysitting is less than it was. The cafe is still open.

References

[1] Luc, S. *Misfit Circuit* - public project notes, including the documented review run of Sept 26. misfitcircuit.com.

[2] Luc, S. *Stephen's Laws* - operating laws of the crew (specs don't sit; restore before patchwork; the four parts of done), codified 2026-10-26/28, crew distribution.

[3] Composer owned worklist and WORKLIST.md - the single owned inventory of the factory's work, incl. the north-star framing.

[4] Plug-in system package (FULL, 2026-10-27): PLUGIN-SPEC and normative SPEC - receipt contract, reason vocabulary, redaction, §15 deliberate omissions. SHA-256 ae34064c…08083.

[5] Provider plug-in integration candidate (2026-10-28): CURRENT-STATE-REVIEW (with 8 numbered differences against the migration notes), BUILD-REPORT, INSTALL-FOR-FAYE. SHA-256 3e6ca956…f74d3.

[6] Composer workflow candidate + fix batch 1 (2026-10-28): build/fix reports, full hashes 442fb6f5… and b0dc5862…; crew install record, composer-workflow-install-20261028/INSTALL-RESULTS.md (Mac-side).

[7] Capture system specification and certification (2026-10-28) - adjacent system; the circle of capture, grading, and case compilation referenced in §2.

[8] Ops Board / Desk architecture notes (2026-10-28) - the results tray, derived board, and fail-closed closure linkage discussed in §7.4.

Author's note. The author thanks the crew - Chad, Command, Way, Faye, Rowan, and the wider roster - whose receipts made this paper possible and whose findings made it shorter in the claims than in the writing. Correspondence to the author. The factory's response time is best-effort; the factory is, at the time of writing, busy running itself.