Institutional Requirements for Sovereign Local Agent Runtimes

Wulf A. Kaal, Ph.D.[1]

[1] Professor of Law, University of St. Thomas School of Law

Provenance. This Article extends a research program on decentralized governance, reputation systems, and dynamic regulation developed across 132 published works over two decades. The framework applied here derives entirely from that published record: the empirical documentation of the institutional deficit in decentralized organizations (Kaal 2026f), the architecture and controlled evaluation of a reputation substrate for autonomous agent populations (Kaal 2026g; Kaal 2026d), its governance treatment (Kaal 2026e), and the economic analysis of computational abundance (Kaal 2026b; Kaal 2026c; Kaal 2026h). The corpus is published at claim resolution with per-work content hashes, so the date and provenance of every concept employed here can be verified against the published record rather than taken on the author's word.

Basis and method. The architectural description in Part III and every finding in Part VII derive from direct reading of the Apache-licensed *mosaic-companion* repository at commit 2d920ceff367ff7a2c73cbf04333572fca465182, pushed July 31, 2026 and retrieved August 18, 2026, whose application and shared layers comprise approximately 20,800 lines of TypeScript across 85 modules, alongside fourteen architecture documents. The commit rather than the branch is cited because the project is under active development and any finding may be remediated after the retrieval date. Negative findings, meaning assertions that a capability is absent, were established by repository-wide search and can be reproduced by any reader against the same commit. The analysis covers the public repository only. Private branches, unreleased work, and the hosted beta service were not examined and no claim is made about them.

Independent derivation. Every requirement, evaluation criterion, and institutional concept in this Article derives from the author's prior published corpus. No confidential, proprietary, or otherwise non-public information of any party was used or relied upon in any part of this Article. The sole third-party source is the publicly available repository cited above, released under the Apache License 2.0. Nothing in this Article derives from private communications with any party, and nothing in it discloses information subject to any confidentiality obligation of the author.

Nature and scope of claims. This Article is a requirements paper with a conformance analysis attached. It is not an empirical study, and it reports no new experimental data; the empirical warrant for its institutional claims rests on the author's prior published empirical work (Kaal 2026f; Kaal 2026d). It distinguishes throughout between what a system's documentation specifies, what its shipped code implements, what this Article requires, and what this Article proposes as remedy. Requirements are stated with evidence clauses so that compliance can be tested by a third party rather than asserted by the author. The engineering contributions in Part VIII are minimal modifications to an existing design, not an architecture; the reputation computation, evaluation methodology, value-attribution algorithm, and contestation design referenced by Requirements 4 through 6 are developed in the author's separately published works (Kaal 2026g; Kaal 2026e) and are not specified here.

Status as scholarship; reserved rights. This Article is a scholarly work and nothing else. It is not a requirement specification, statement of work, deliverable, or work product prepared for or at the request of any party, and it was not commissioned. Its publication conveys no license, assignment, or other right in any mechanism, architecture, or method developed in the author's corpus beyond the rights ordinarily conveyed by scholarly publication, and no commercial party may treat this Article as a specification against which work is performed or owed.

Not a security audit. This Article is an institutional and design analysis. It is not a security assessment, and no penetration testing, fuzzing, adversarial probing, or vulnerability research of any kind was performed. The findings concern accountability, provenance, and record-keeping properties of a design, not exploitable vulnerabilities, and no finding should be read as a vulnerability disclosure or as a representation about the security posture of any deployment. Readers requiring a security assessment should commission one.

Relationship disclosure. The author has from time to time held commercial discussions with parties developing systems in the class this Article analyzes. No such discussion has resulted in an executed agreement, and the author has received no compensation from, and holds no equity, tokens, or other financial interest in, any such party. No commercial party commissioned, funded, reviewed, or approved this Article, and none exercised editorial control over it. The author discloses this so that readers may weigh it; readers should do so.

Abstract

A new class of system has arrived. Models, agents, tools, and data resources now compose on the user's own machine, hold state locally, and settle payment for outward work machine-to-machine. This Article calls that class the sovereign local agent runtime and argues that its central achievement is narrower than its advocates claim. Sovereignty is a custody property. Accountability is an institutional property. The first does not produce the second. Relocating execution to the user's device answers the question of who holds the data. It does not answer who acted, under what authority, over which data, toward what outcome, and with what recourse when the outcome is wrong. Those are institutional questions, and architecture alone does not answer them. The Article specifies six institutional requirements that any adequate coordination layer for such runtimes must satisfy, derives each from two decades of research on decentralized governance, reputation, and dynamic regulation, and applies them to the shipping public implementation of Mosaic, read from source, as a worked example. That reading produces eight findings, of which the most consequential is that cross-component provenance is not merely incomplete but structurally impossible in the Chronicle: records are per-tool and carry no workflow, causal, or principal identifier, so a workflow spanning three tools produces three logs that cannot be joined, while the built-in, protocol-server, and payment execution paths produce no records at all. The Article closes with six minimal-delta engineering contributions, each expressed against the existing design and none requiring a new subsystem. The first two convert cross-component correlation from impossible to feasible and make local alteration detectable. Full satisfaction of the provenance requirement additionally demands coverage of every execution path and evidence a third party can verify.

Conflict-of-interest disclosure. The author is simultaneously the theorist, the protocol architect, and the empiricist for the research program developed in the cited corpus works. That concentration of roles is a limitation readers should weigh in evaluating any claim about the architecture developed in the cited corpus works. This Article was produced on compute infrastructure owned by the author. No external funding supported this research.

Status. Working draft, August 2026. This Article has not been peer reviewed and remains subject to revision. Comments welcome at wulf@wulfkaal.com.

Table of Contents

I. Introduction

The hard problem moved and the field did not follow it.

For a decade the governing difficulty in distributed computation was custody. Who holds the data, who holds the keys, who can be compelled to produce them, and what happens to a user whose provider fails, is acquired, or changes its terms. That problem is now substantially solved at the level of architecture. A user can run models locally, keep state on hardware under personal control, compose tools without surrendering credentials to an intermediary, and pay for outward compute in increments too small for a payment processor to intermediate.

The achievement is real and it is also narrower than the language surrounding it suggests. Sovereignty, as this Article uses the term, is a property of custody. It says where a thing sits and who may reach it. Accountability is a property of institutions. It says who acted, under what authority, over which data, toward what outcome, and what follows when the outcome is wrong. The first does not produce the second. A user who holds every byte locally and composes fifteen services into a workflow has perfect custody and no answer at all to the question of what happened.

This Article names the class of system in which that gap appears: the sovereign local agent runtime. The naming matters. Analysis pinned to a product expires when the product does. Analysis pinned to a class of systems survives, and the class is now populated well enough to study. Mosaic serves throughout as the worked example, described only from its public architecture, because it is a serious and unusually complete instance of the class rather than because it is the only one.

The argument proceeds in six movements. The Article first defines the class by four testable properties rather than by vendor or by branding. It then argues that connectivity between heterogeneous components is not coordination among them, and that the difference is measured in transaction costs that throughput benchmarks cannot see. It then states the accountability gap precisely, resolving what is usually discussed as a general anxiety about trust into four independently diagnosable failure conditions: attribution failure, authority drift, evaluative capture, and recourse vacuum. It then specifies six institutional requirements that any adequate coordination layer must satisfy, and states for each what evidence would show the requirement met. It then reads one running implementation against those requirements. Finally it

proposes the minimum changes that would satisfy them.

The Article is deliberately normative rather than constructive. It states what a solution must achieve and how compliance would be tested before it proposes any remedy, and the remedies it does propose are minimal modifications to one existing system rather than an architecture of its own. That sequence is methodological rather than coy. The history of governance in decentralized systems is substantially a history of implementations that shipped before their requirements were stated, and the resulting institutional deficit has been argued at length on an empirical record (Kaal 2026f). Requirements stated in advance are testable. Implementations shipped in advance of requirements are merely present.

The claim that unifies the Article is that reputation, properly engineered, is the load-bearing institution for this class of system, and that it fails precisely when it is assumed rather than engineered. That claim has a long provenance in the literature developed here (Calcaterra, Kaal, and Andrei 2018; Calcaterra and Kaal 2021; Kaal 2024b), and its extension to autonomous agent populations has been worked out separately (Kaal 2026e). This Article supplies what those treatments presuppose: the conditions under which a reputation layer may be introduced into a sovereign runtime without becoming a security credential, a marketing surface, or a decorative score.

II. The Sovereign Local Agent Runtime as a Class of System

A class definition should be testable. Membership in the class defended here turns on four properties, each of which can be checked against a running system rather than against a description of one.

First, execution occurs on hardware the user controls. The relevant test is not where a company says computation happens but whether the user can disconnect the machine from the network and retain the capability. Systems that degrade to nothing when isolated are remote services with a local presentation layer.

Second, state, credentials, and data resources are user-held rather than platform-held. The test is whether the operator can read the user's working state without the user's participation. A system in which the operator holds decryption capability holds the data, whatever the marketing says about ownership.

Third, composition is heterogeneous. Models, agents, locally installed tools, external services, and networked compute participate in a single workflow. This property is what distinguishes the class from a local application. A local word processor is sovereign and uninteresting for present purposes. A local runtime that recruits an external model, invokes three tools, consults a remote node, and pays for the privilege is sovereign and institutionally novel.

Fourth, settlement for outward work occurs machine-to-machine. Payment for compute, inference, or service occurs between components without a platform taking position in the middle. This property converts a technical composition into an economic one and brings with it the entire apparatus of contract, attribution, and dispute that economic relations require.

Mosaic implements all four properties in its public code, the fourth through a payment plugin that intercepts payment-required responses from network nodes and settles in tokens autonomously under per-transaction and daily policy caps. Because the

argument that follows depends on what the system actually does rather than on how it is described, Part III reads that implementation out of the public repository directly.

Two clarifications prevent the class from expanding until it is useless. The class does not include systems whose local component is a cache or an accelerator for a remote service, because the disconnection test fails. It does not include local applications with plugin architectures, because heterogeneous composition in the sense used here requires that the composed components be independently operated, independently incentivized, and capable of failing in ways the runtime did not anticipate. A plugin written by the vendor is a feature. A service invoked across an economic boundary is a counterparty.

The distinction between a feature and a counterparty is the entire subject of this Article. Features are governed by their vendor's own accountability. Counterparties require institutions between parties.

III. The Mosaic Architecture as Built

This Part describes Mosaic from its public implementation rather than from its marketing. All statements are taken from the Apache-licensed repository *mosaic-companion* at commit 2d920ce (pushed July 31, 2026; retrieved August 18, 2026), whose application and shared layers comprise roughly 20,800 lines of TypeScript across 85 modules, alongside fourteen architecture documents. Where the documentation and the code disagree, the code governs, and the disagreements are noted because they are informative.

A. The two-zone trust model

Mosaic partitions itself into a trusted Core and an untrusted Sandbox. Core owns policy control, user approvals, secrets, audit logging, storage coordination, boundary enforcement, the outbound Gatekeeper, the user wallet, and the container launcher. The Sandbox runs tool modules, third-party Model Context Protocol servers, and, prospectively, agents.

The governing invariant is stated in the repository in terms this Article endorses without reservation: "Tool execution is always low-trust. Tools do not gain trust because they run in containers managed by Mosaic," and, separately, "Containers are NOT the security boundary. Core enforcement is." The design further holds that "Trust is architectural, not reputational."

That last sentence deserves emphasis, because it anticipates Requirement 1 below and settles it correctly in advance. The Mosaic authors have already concluded that reputation must not be load-bearing for access control. The requirement stated later in this Article is therefore not a correction. It is a formalization of a commitment the implementation has already made, together with a test that would show the commitment held as a reputation layer is introduced.

Every boundary crossing must be explicit, core-mediated, and logged. The architecture deliberately does not mandate a topology: one container per tool, several tools per container, in-process WASM modules, and child processes are all admissible, provided Core mediation, policy, and logging semantics hold regardless.

B. What actually ships

The implementation status document and the code diverge from the overview in one material respect. The overview describes Docker containers as the execution substrate and concedes candidly that "Docker IS a hard runtime dependency for v1." The shipping runtime, however, is WASM-first: an Extism BackgroundPlugin worker with host functions, tool artifacts persisted to disk, manifests embedded in the binary and extracted through a mosaic_manifest() export. Both statements are true of different layers, but a reader of the overview alone would form the wrong picture of what runs today.

Shipping subsystems include the WASM tool sandbox with install-time approval and permission diffing on update; the Gatekeeper domain allowlist with audit logging; the per-tool Chronicle; the Vault with box-level agent access control; a tool registry exposing built-in Gmail, Web3, and Vault modules alongside MCP servers and WASM tools; a multi-provider agent system with a recursive tool-use loop; and a just-in-time payment plugin that intercepts payment-required responses from network nodes and transfers tokens autonomously under per-transaction and daily caps.

C. The manifest as contract

The manifest is the declared contract between a tool developer and the runtime. It carries identity, runtime entry, permissions, resource limits, inputs, exposed functions, and UI panel declarations. Permissions in version 1.0.0 comprise internet, allowed_domains, files, and services. The documentation additionally describes CPU, disk, and GPU allocations, but the shipping ToolResources type carries only a memory limit and a per-call timeout.

Three properties of the manifest matter for the argument. Permissions are declared in advance and approved by the user before installation. There is no runtime escalation: a tool cannot request additional authority mid-execution, and a version update requesting new permissions triggers a diff and re-approval. And the manifest is embedded in the artifact rather than shipped beside it, which means the declaration and the code are bound together.

D. Artifact integrity

Mosaic computes the SHA-256 of a tool artifact at install and verifies it again before every launch. If the stored file was modified after approval, launch fails with an integrity error. Legacy installations without a recorded hash are backfilled on first launch.

This is the strongest primitive in the system for present purposes, and Part VIII argues it is currently underused. A content hash that gates execution is a durable identifier for an exact artifact version. It is the anchoring primitive a reputation layer requires and rarely has, and Part VIII is careful about what it can and cannot carry.

E. The Chronicle

The Chronicle is the only channel through which a tool may write. It is a per-tool

append-only JSONL file under the application's user-data directory, at chronicles/<tool_id>/chronicle.jsonl, written by Core on the tool's behalf through host functions. Entries carry an identifier, an ISO timestamp, a source (tool, gatekeeper, or core), a type (log, output, audit, lifecycle), and a structured data payload. Gatekeeper decisions are written into the same stream automatically, which means policy outcomes and tool behavior share one ordered record.
The design rationale recorded in the repository is unusually complete: audit trail, debugging by replay, security review, behavioral data mining, and eventual state reconstruction of a killed container from its own history.

F. The Gatekeeper

All outbound traffic from the Sandbox passes through Core. The implemented policy is a per-tool exact-match domain allowlist drawn from the manifest, with normalization to lowercase and trimming, plus a parallel file-path check. Denials carry a reason string, and both allow and deny decisions are recorded as audit entries. One caveat, developed in Part VII: a tool that uses the runtime's built-in HTTP facility is domain-restricted at plugin construction rather than per request, and that path is not shown to emit Gatekeeper audit entries.
The repository is explicit that network isolation alone is insufficient, because Docker networks cannot filter by domain, inspect content, detect personal information, log destinations, or apply per-tool policy. The Gatekeeper exists to supply exactly those capabilities. Part VII records which of them exist in code today.

IV. Connectivity Is Not Coordination

Composition without common rules multiplies transaction costs rather than reducing them.
The New Institutional Economics tradition supplies the analytic frame (Coase 1937; Williamson 1985; North 1990), and the application of that frame to fast-moving technological domains has been developed at length in the dynamic regulation literature (Kaal 2013b; Kaal 2016). The relevant insight is Coasean. The boundary of an organization is set by the relative cost of transacting inside it versus outside it. Where the cost of specifying, monitoring, and enforcing an external relation is high, activity migrates inside the firm. Where it is low, activity disperses into markets.
A sovereign local agent runtime inverts the usual geography of that calculation. It disperses activity to the maximum possible degree, placing the boundary at the individual device, and then asks a user or an agent to transact across that boundary continuously and at machine speed. The architecture assumes the transaction cost of external relation is near zero. It is not. It has merely been made invisible.
Consider what each new integration in such a runtime actually requires the parties to settle. They must settle identity, meaning what a component is and how a claim to be that component is tested. They must settle authority, meaning what the component is permitted to do and on whose behalf. They must settle permitted data use, meaning what may cross the boundary, for what purpose, and for how long. They must settle responsibility, meaning who answers for an outcome produced jointly. They must settle remedy, meaning what happens when the answer is unsatisfactory.

A runtime can mediate some of this centrally, and the better ones do. A trusted core that brokers every crossing supplies a shared answer to identity, to authority, and to permitted data use, and to that extent it converts what would otherwise be pairwise negotiation into a single contract each component signs once. That is the standard and correct fix, and it is why transport and policy mediation should not be expected to scale badly. What such a core does not supply is a shared answer to responsibility, to remedy, or to value attribution, because those are settled between the parties to a particular outcome rather than at the boundary of a particular call. In the absence of a common institutional layer, those three remain bilateral and idiosyncratic, and their cost, on the analysis offered here and marked for measurement in Part XI, grows faster than linearly in the number of components, borne in integration effort, in security review, in support burden, and eventually in the user's inability to determine what occurred. This is the true constraint on scale for the class, and it is invisible in every benchmark the field currently reports. Throughput, latency, and token cost measure the performance of composition. They do not measure the cost of arranging it.

The economic framework required to reason about this regime has been developed under the heading of Computative Economics, which treats computational abundance as a condition that relocates scarcity rather than eliminating it (Kaal 2026b; Kaal 2026c; Kaal 2026h, the last two being distinct statements of the framework). Where compute is abundant and coordination is scarce, the returns accrue to whoever supplies coordination. The operational counterpart, describing how agent coordination systems generate and traverse a possibility space, is developed separately (Kaal 2026i). The present Article takes both as given and asks the narrower institutional question that follows from them.

That question is this. If coordination is the scarce good, what must a coordination layer accomplish before it deserves the name?

V. The Accountability Gap, Stated Precisely

The gap is not vagueness about trust. Vagueness about trust is a symptom. The condition resolves into four failure conditions, each independently diagnosable and each with a distinct remedy.

A. Attribution failure

In a workflow crossing several components, no participant can establish which component produced which part of the outcome. A user receives an answer assembled from a local model, a remote inference call, two tool invocations, and a retrieval step. The answer is wrong in a specific way. Nothing in the system supports the question of where it went wrong.

Attribution failure is not merely an inconvenience for debugging. It is the dissolution of responsibility into composition. Where contribution cannot be traced, fault cannot be assigned, and where fault cannot be assigned, no participant has an incentive to prevent it. The agency-cost structure this produces has been analyzed in the corporate context (Kaal 2019b) and measured directly in multi-model agent cohorts

(Kaal 2026d).

B. Authority drift

Delegation outlives the consent that created it. A user grants a component access to a data resource for a stated purpose. The component delegates onward. The onward component caches, or persists, or is itself invoked later by a different workflow under the same grant. The original consent was bounded in purpose and time. The operative authority is bounded by neither.

Authority drift is the failure mode that permissions systems are least equipped to detect, because at every individual step the system behaves correctly. Each check passes. The aggregate is nonetheless outside what the user authorized. Revocation compounds the problem: withdrawal of a grant at the point of issue does not reliably propagate to components already operating under it, and few systems can demonstrate that it did.

C. Evaluative capture

Selection among services depends on signals that the selected party controls. Self-reported capability, static ratings, installation counts, and volume metrics are each manipulable, and none is contextual. A component that performs well on one class of task and poorly on another carries a single undifferentiated score, if it carries one at all.

The consequence is that the market for services clears on claims rather than on outcomes. This is a well-documented failure in decentralized commerce generally (Kaal 2019c) and the specific reason that verified reputation was proposed as a precondition for distributed task markets rather than as an enhancement to them (Kaal 2018b). The manipulation surface is not hypothetical. Sybil creation, collusion, wash interaction, and reputation laundering across contexts are the standard attacks, and a scoring system that does not name its resistance to each of them has not specified a threat model.

D. Recourse vacuum

When an outcome is wrong, no defined path exists to contest it, and no party holds an obligation to answer. This is the failure condition that distinguishes an institution from a mechanism. A mechanism produces outputs. An institution produces outputs and also produces a way to challenge them.

The recourse vacuum is the least discussed of the four and the most consequential for adoption by regulated entities. An enterprise struggles to deploy a system in which no participant is answerable, not because the technology is inadequate but because the arrangement resists insurance, indemnification, and internal approval, and must be compensated for by contractual and organizational controls the runtime does not supply. The institutional deficit that this produces in decentralized organizations has been documented empirically and is not a matter of opinion (Kaal 2026f; Kaal 2020).

VI. Six Institutional Requirements

What follows is the normative core of the Article. Each requirement states what an adequate coordination layer must achieve and what evidence would show that it has been achieved. None states how to build one. The set is a minimum, not a taxonomy: organizational accountability, named risk ownership, human oversight, incident handling, and lifecycle governance are presupposed complements that no runtime record can replace. The distinction is load-bearing: a requirement that can only be satisfied one way is a specification wearing a disguise, and specifications foreclose the design search that this field still needs.

Requirement 1: Separation of reputation from authorization

Reputation may inform discovery, routing, pricing, and allocation. It may never grant permission, enlarge authority, or substitute for an enforcement boundary.

This is the first requirement because violating it destroys the others. A system that permits standing to open a door has converted a performance signal into a security credential, and every incentive to manipulate the signal now carries the payoff of privilege escalation. Reputation systems are robust when the cost of manipulation exceeds its benefit. Attaching authorization to reputation raises the benefit without raising the cost.

Evidence of compliance: a demonstration that a maximally reputable component and a minimally reputable component encounter identical enforcement at every protected boundary, differing only in whether they are selected to attempt the crossing.

Requirement 2: Provenance that survives composition

The record of who acted, under what authority, and over which data must remain intelligible after the workflow completes, after intermediate components are replaced, and after the composing runtime has moved on.

Provenance that exists only inside a live session is telemetry. Provenance in the sense required here must be durable, portable, and interpretable by a party who did not observe the original execution. That third property is the difficult one and the one most often omitted.

Evidence of compliance: reconstruction of a completed multi-component workflow by an auditor with no access to the runtime that executed it, sufficient to identify each participant, the authority each exercised, and the data that crossed each boundary.

Requirement 3: Consent that revokes downward

Withdrawal of authority must propagate to every component operating under it, and the system must be able to demonstrate that propagation occurred.

The demonstration clause is the substance of the requirement. Systems commonly claim revocation and implement invalidation at the point of issue only. The user experience of revocation is then indistinguishable from its absence for any component that has already cached the grant or derived a secondary credential from it.

Evidence of compliance: a revocation event followed by an attempted downstream

action under the revoked authority, with the refusal recorded at the enforcement boundary rather than at the point of issue.

Requirement 4: Outcome-based evaluation resistant to self-report

Evaluation must derive from observed results under stated conditions, must be specific to context, and must be costly to manipulate.
Each clause excludes a common practice. Observed results excludes self-declared capability. Stated conditions excludes aggregate scores that average across incommensurable tasks. Costly to manipulate excludes any signal a participant can generate at will, which in practice excludes most engagement metrics.
Evidence of compliance: a stated threat model naming Sybil creation, collusion, wash interaction, and cross-context laundering, together with the cost imposed on each and the measurement showing that cost exceeds the achievable gain.

Requirement 5: Attribution of value across multi-party workflows

Contribution must be traceable well enough to support payment, credit, or recurring compensation without a central assignor.
This requirement follows from the fourth defining property of the class. Once settlement is machine-to-machine, value allocation is a coordination problem rather than an accounting one, and a runtime that routes work without allocating value has externalized the hardest part of its own economics onto its participants. The relevant precedents include token model design across a large comparative sample (Kaal 2018a) and the treatment of reputation itself as a capital asset rather than as a score (Kaal 2021b; Kaal 2021c).
Evidence of compliance: a completed workflow in which each contributing component's share is derivable from the provenance record alone, without reference to a privileged ledger held by the runtime operator.

Requirement 6: Contestability

A defined path must exist to challenge an outcome, and the path must terminate in a remedy rather than in an explanation.
The terminal clause distinguishes this requirement from transparency. Explanation is valuable and insufficient. An institution is constituted by the consequences it can impose, which is why engineered consequence rather than disclosure is the operative design principle (Kaal 2026a). Governance arrangements that produce reasons but not remedies are advisory, and advisory governance is the characteristic failure of decentralized organizations at scale (Kaal and Bykowski 2023; Kaal 2024d).
Evidence of compliance: a documented instance in which a challenge altered an allocation, a standing, or an authority, together with the elapsed time from challenge to effect.

VII. Findings: The Implementation Against the Requirements

The findings below are offered as a build agenda for a system that has already made the hardest architectural choice correctly. Mosaic committed to Core-mediated enforcement and to treating tool execution as low-trust regardless of provenance, and that commitment is what makes the remaining gaps worth closing rather than worth abandoning. What follows evaluates the shipping implementation against the six requirements. Each finding is grounded in a specific file or document in the repository at commit 2d920ce. Findings are stated as defects of the current state, not as criticisms of intent. In several cases the implementation is closer to the requirement than any comparable system. The gaps divide in kind: some are fields not yet present on records that exist, and nothing in the design forecloses adding them; others are write paths that do not exist, and those produce no record to which a field could be added.

Finding 1 (Requirement 1): Satisfied in principle, untested in fact

The architecture already commits to the separation this Article requires. Trust is architectural rather than reputational, and enforcement is Core-mediated regardless of what a component claims about itself. No reputation input reaches the enforcement path because no reputation layer exists yet.

The requirement is therefore not violated. It is unguarded. Nothing in the codebase would prevent a future reputation score from being consulted in ManifestGatekeeperPolicy.checkDomain, and the pressure to do so will arrive the moment a reputation layer exists and someone proposes that highly reputable tools should face fewer prompts. The evidence clause in Requirement 1 should be added as a regression test now, while it is trivially true, rather than later when it is contested.

Finding 2 (Requirement 2): End-to-end provenance is not producible from the audit surfaces

This is the most consequential finding in the Article.

The Chronicle is per-tool by design: each tool writes to its own file, and the documentation states that no tool can see another tool's Chronicle. The entry structure, defined in electron/integrations/sandbox/types.ts, is exactly five fields: id, timestamp, source, type, and data. There is no workflow identifier, no parent-entry reference, no agent or principal identity, and no session or correlation field on the entry. Session identifiers exist in the agent chat interface; they are not written onto Chronicle records and cannot join them. A repository-wide search for trace, workflow, correlation, parent, or session identifiers in the sandbox integration returns nothing.

The consequence is precise. When an agent composes three tools into a single user-facing answer, the Chronicle produces three disjoint append-only logs with no

field that relates them. Chat history, if it records tool-call order at all, is a deletable conversation transcript rather than a join key, and the only correlation available from the Chronicle is timestamp proximity, which is unreliable under concurrency and unusable as evidence. Attribution failure, described in Part V as a general hazard of the class, is therefore not a latent risk in the Chronicle as specified. It is the current behavior of the record.

The defect is broader than joinability. The Chronicle write path covers only sandboxed tools, and only what those tools explicitly write. The built-in Gmail, Web3, and Vault modules and third-party MCP servers execute through the tool registry with no Chronicle instrumentation at all. A sandboxed tool's ordinary return value is not recorded unless the tool itself calls the output host function. A tool that uses the runtime's built-in HTTP facility is domain-restricted by the runtime but bypasses the audited host function, so its outbound traffic produces no Gatekeeper audit entries. And the payment plugin, which moves real value autonomously, writes nothing into any Chronicle. End-to-end provenance is therefore not merely unjoinable across the records that exist; for several execution and network paths, no record exists to join.

Table 1 states the coverage path by path. The first two rows describe the sandboxed path: what a tool writes through the host functions is recorded, and what it merely returns is not. The remaining rows are the paths that carry real work and, in the payment case, real value, while writing nothing an auditor could later read. Cells marked not addressed record the limit of the finding rather than a conclusion, and every absence claim is reproducible by repository-wide search at the cited commit.

Table 1. Audit-surface coverage by execution path at commit 2d920ce.

Execution path	Chronicle record	Gatekeeper audit entry
Sandboxed WASM tool, explicit host-function writes	Yes: log and output entries via the write host functions	Yes, for calls through the audited host functions
Sandboxed WASM tool, ordinary return value	No: returned to the agent, not recorded	n/a
Built-in modules (Gmail, Web3, Vault)	No: no Chronicle instrumentation in the tool registry	Not addressed
MCP servers	No: no Chronicle instrumentation	Not addressed
Extism built-in HTTP (allowedHosts)	Not addressed	Not shown to emit: domain-restricted at plugin construction, bypasses the audited host function
Payment plugin (payments-jit)	No: writes nothing into any Chronicle	Not addressed

Note: Not addressed marks paths whose behavior the finding does not establish; not shown to emit follows the formulation in Part III.F. Source: author's analysis of the Apache-licensed mosaic-companion repository at commit 2d920ce, retrieved August 18, 2026.

Two secondary defects compound it. Entry identifiers are generated as Date.now() concatenated with a short Math.random() suffix, which is not a monotonic sequence identifier, not unique by construction, and does not establish ordering across files. And the runtime's own open engineering questions record that the Extism BackgroundPlugin is not reentrant, listing a sequential queue as one planned option; the log records no execution order, so temporal inference from timestamps cannot settle concurrency the implementation has not yet settled.

Figure 1 assembles the finding from the auditor's side of the boundary. The three files at the base are the only records the workflow leaves. Each carries the five fields of the shipped entry type and nothing that names the workflow, the parent call, or the principal. The chat session above them is the one place a session identifier exists, and it is a deletable transcript rather than a join key. The paths on the right execute without writing at all.

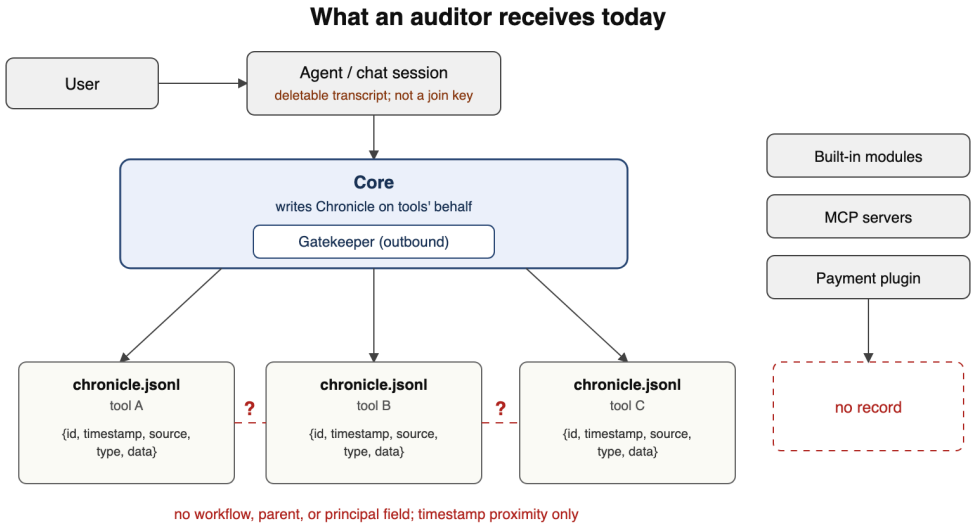

Figure 1. What an auditor receives today: Chronicle records are per-tool and carry no join field, and several execution paths write no record at all. Entry fields from electron/integrations/sandbox/types.ts at commit 2d920ce; author's rendering of Finding 2.

Finding 3 (Requirement 2, continued): Append-only is structural, not verifiable

The repository states plainly that "Enforcement is structural in v1": no update or delete API exists, and tools can only reach the log through Core's append path. Within the process boundary this is sound.

It is not tamper-evident. The Chronicle is a plain JSONL file in the user's configuration directory. Any process with filesystem access, including a compromised host application or the user, can rewrite or truncate history, and nothing in the record would reveal it. Entries carry no digest, no chaining to a predecessor, and no periodic root.

The system verifies the integrity of tool artifacts with SHA-256 while leaving the record of what those artifacts did unprotected.

Finding 4 (Requirement 2, continued): The reader silently discards evidence

Two behaviors in Chronicle.read defeat the audit purpose the Chronicle exists to serve. Malformed lines are skipped with a console warning and omitted from the returned entries, so a corrupted or tampered record disappears from view rather than raising. And the query applies a default limit of one hundred entries, returning entries.slice(-limit), so a caller who does not specify a limit silently receives the most recent hundred entries with no indication that anything preceded them.
An audit log that truncates without saying so, and drops unparseable evidence without escalating, reports a clean history in exactly the circumstances where the history is not clean.

Finding 5 (Requirement 3): No revocation propagation, and a live authority gap

Vault box access is enforced at runtime through an execution context carrying the agent identity. Withdrawal of a grant therefore stops future reads at the point of issue. Nothing propagates to data already materialized into a tool's inputs or already written into a Chronicle, and no refusal under a revoked authority is recorded at the enforcement boundary.
A distinct and larger gap sits alongside it. The permission model records that per-agent tool access and per-agent internet access are not yet implemented, and that consequently all tools are available to all agents. Granting an agent access to a single Vault box therefore grants it the ability to invoke every installed tool, whose union of declared domains defines the true outbound surface. The user approves permissions tool by tool and receives their composition without ever seeing it.

Finding 6 (Requirement 4): Documented filtering layers are absent from the code

The Gatekeeper documentation specifies four filtering layers: domain allowlist, content and MIME type inspection, a personal-information baseline using regular expressions and named-entity recognition, and audit logging. The implementation contains the first and the fourth. A search of the sandbox integration for personal-information, MIME, or redaction handling returns nothing.
This is reported as a documentation defect rather than a security failure, because the documentation is aspirational in tone and the implementation status does not claim the filters as complete. It matters nonetheless: an enterprise reading the architecture documents would reasonably conclude that outbound content inspection exists today, and it does not.
No outcome-based evaluation layer exists, which is expected at this stage and is the subject of Part VIII.

Finding 7 (Requirement 5): No economic substrate in the contract layer

The class of system is defined in part by machine-to-machine settlement. The manifest, which is the only contract a tool presents, contains no economic field whatsoever: no price, no settlement address, no attribution declaration, no revenue expectation. The wallet is user-held: the runtime creates one, the user funds it, and an existing private key can be imported through a dedicated secure-import window. Tools have no wallet of their own, and agent wallets are recorded as a future capability.
Value already moves: the payment plugin intercepts a payment-required response, transfers tokens under policy caps, and retries the call without human involvement. Yet nothing in the contract or record layer can express which component's contribution earned what moved, and the payment path itself writes no Chronicle record. Requirement 5 has no substrate to attach to, and this is the gap most likely to become expensive later, because pricing conventions harden quickly once an ecosystem has participants.

Finding 8 (Requirement 6): No contestation path

No challenge procedure appears in the documentation or the code. Gatekeeper denials produce a reason string and an audit entry, which is an explanation. Nothing converts a user's disagreement with an outcome into an alteration of allocation, standing, or authority. Contestability is the requirement least often implemented in any system of this kind, and it is the one that most directly determines whether regulated entities can deploy the system.

VIII. Engineering Contributions

Status of this Part. Part VI states the requirements. This Part is illustrative: it shows the minimum modifications that would satisfy those requirements in one public snapshot of one system. It is not a general architecture, not a requirement specification for any project, and not work product prepared for or licensed to any party.
The findings above are actionable, and the actions are small. This Part proposes six changes, ordered by ratio of institutional value to implementation cost. Each is expressed as a delta against the existing design rather than as an architecture, each preserves the two-zone trust model and the low-trust treatment of tools, and none requires a new subsystem. The first two together convert cross-component correlation from impossible to feasible and make local alteration detectable; what full satisfaction of Requirement 2 additionally requires, coverage of every execution path and externally verifiable completeness, is stated inside each contribution rather than assumed away. The same candor applies to Requirement 3: Contribution 5 produces its evidence-clause artifact, while graph-wide demonstration of propagation remains outside a minimal delta.

Contribution 1: Trace context on Chronicle entries

Extend the entry structure with three optional fields: a workflow identifier constant across every component participating in one user-initiated task, a parent-entry reference giving causal order within that workflow, and a principal identifier naming the agent or user on whose authority the action was taken.
Core mediates every boundary crossing and is therefore positioned to mint and thread all three values. Today it does not carry them: the execution context threaded through the tool registry holds only an optional agent identity, and the bridge that adapts sandboxed tools does not receive or forward even that, so the values must be generated and propagated through the call path rather than merely copied into the record. The same fields must also be written by every execution path, including the built-in modules, protocol servers, and the payment plugin, or the provenance layer inherits the coverage gap in Finding 2. Nothing needs to be surfaced to tools, which is important: the tool remains low-trust and cannot forge its own provenance, because it never supplies these fields. The change is additive, backward compatible with existing logs, and makes cross-tool reconstruction a matter of selection rather than inference.
Among components that write into the Chronicle, the audit trail can then answer which component produced a wrong result; paths that write nothing remain silent until instrumented.

Contribution 2: Hash chaining for tamper evidence

Give each entry a digest over its own content and its predecessor's digest, and persist the current head digest per tool. Append cost is one hash. Verification is a linear scan. Any modification, deletion, or reordering of history breaks the chain at a determinable point.
The system already computes SHA-256 over tool artifacts and refuses to launch on mismatch, so the primitive, the dependency, and the failure convention all exist. Extending the same discipline from the artifact to the record closes the asymmetry in which Mosaic verifies what a tool *is* but not what it *did*.
A periodic root across all per-tool chains, published or merely retained, would additionally allow a third party to verify that a presented log is the one that was written, which is the property Part X depends upon.

Figure 2 shows the state Contributions 1 and 2 produce, and what they deliberately do not. The three files now share a workflow identifier minted and threaded by Core, so reconstruction is selection rather than inference. Each file chains entry to entry into a head, and the heads feed a periodic root that a third party can check. The panel on the right is unchanged from Figure 1: until instrumented, the built-in, protocol-server, and payment paths still write nothing.

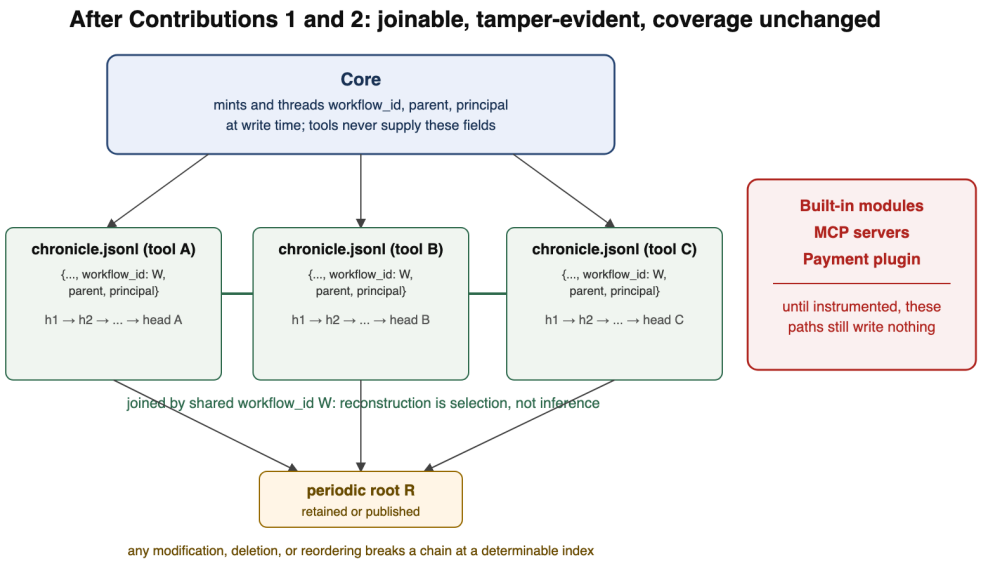

Figure 2. After Contributions 1 and 2: correlation becomes selection and local alteration becomes detectable. Uninstrumented paths remain the stated remainder. Author's rendering.

Contribution 3: Fail loudly on read

Return truncation and parse failures as part of the result rather than discarding them. A read that returns one hundred of four thousand entries should say so; a malformed line should surface as an explicit gap marker at its position rather than vanish. Neither behavior is a design decision to defend; both are ordinary defaults that happen to be wrong for an audit log.

Contribution 4: Bind reputation to the artifact hash

When a reputation layer is introduced, anchor standing to the SHA-256 of the tool artifact rather than to the tool identifier or the publisher name.

The hash is a version-specific subject identifier, not an identity for the developer or operator behind the artifact, and the binding must be stated with care. What it buys is that standing cannot silently transfer across a substantive update, since the hash changes when the code does, and that observed standing attaches to the exact bytes that earned it rather than to a name a publisher controls. What it does not buy on its own is identity cost: a new hash is cheap to mint, which prevents laundering accumulated standing into new code but equally lets an artifact shed a bad record by changing a byte. Hash-bound standing therefore needs a publisher-level link and a controlled migration rule, under which standing carries across versions only by an explicit, recorded act. The identity-cost precondition Part IX names is met by that pairing, not by the hash alone.

The permission diff already shown to users on update becomes the natural place to surface that standing does not carry forward automatically.

Contribution 5: Record revocation at the boundary

When an authority is withdrawn, record the withdrawal, and record subsequent refusals attributable to it at the enforcement point rather than at the point of issue. The Gatekeeper already emits allow and deny decisions with reason strings into the Chronicle, so the mechanism exists and needs only a reason category and the propagation event.

For the refusal to be attributable, the grant itself needs an identifier that derived actions carry, which is the authority half of the trace context in Contribution 1; a denial that cannot name the revoked grant proves only that one enforcement point refused one action. With that identifier, this is close to free and produces the artifact Requirement 3 demands as evidence of compliance at each enforcement point; demonstrating propagation across every component operating under a grant is delegation-graph work beyond a minimal delta, and the Article does not pretend otherwise. It also converts revocation from a claim the system makes into a fact the record shows.

Contribution 6: An economic block in the manifest

Add an optional declaration in which a tool states its settlement address, its pricing basis, and its attribution expectation when composed with others. Declaration is not enforcement, and the proposal deliberately stops short of a settlement mechanism. What it establishes is that the contract layer has somewhere to put the information, so that convention can form around a field rather than around its absence.

The cost of adding an optional field now is a schema version. The cost of adding it after an ecosystem has priced itself informally is a migration.

Table 2 collects the six contributions with the cost each states and the remainder each leaves open. The remainder column is the calibration of this Part in one place: coverage and the root for the provenance pair, the publisher link and migration rule for the reputation anchor, the grant identifier and the delegation graph for revocation, and the reserved attribution mechanism for the economic block. A reader who takes nothing else from this Part should take the pairing of each delta with its stated limit.

Table 2. The six contributions, their stated cost, and the remainder each leaves open.

Contribution	Serves	Cost as stated	What remains
C1 Trace context on Chronicle entries	R2	Three optional fields; additive; backward compatible	Every execution path must write them; paths that write nothing stay silent until instrumented
C2 Hash chaining with periodic root	R2	One hash per append; linear-scan verification	Externally verifiable completeness through the retained or published root
C3 Fail loudly on read	R2	A default-behavior change	None stated
C4 Reputation bound to the artifact hash	Precondition for a reputation layer (Part IX identity cost; the subject of R4's evaluation)	An anchor choice at the introduction of a reputation layer	Publisher-level link and a controlled migration rule
C5 Revocation recorded at the boundary	R3	A reason category and a propagation event on the existing audit path	The grant identifier (authority half of C1); graph-wide propagation beyond a minimal delta
C6 Economic block in the manifest	R5	An optional field; a schema version	Declaration is not settlement; the attribution mechanism is developed elsewhere

Note: Requirement numbers refer to Part VI; remainder language follows Part VIII. Source: this Article, Part VIII.

Sequencing

Contributions 1 and 2 should precede a tool marketplace, because a registry that distributes tools without cross-tool provenance or tamper evidence will accumulate an installed base whose behavior cannot be reconstructed. Contribution 4 should precede any reputation feature, since retrofitting an identity anchor after scores exist requires invalidating them. Contributions 3 and 5 are independent and can land immediately. Contribution 6 should precede the marketplace for the reason given above.

IX. Why Reputation Carries the Load, and Why It Must Be Engineered

Among the institutions that govern counterparty selection, reputation is the one that scales to machine speed while carrying a cumulative record of conduct, and it fails whenever it is assumed rather than engineered.

The first half of that claim is a matter of elimination. Contract requires negotiation and enforcement, both of which are slow and expensive relative to the transaction sizes characteristic of agent interaction. Regulation requires jurisdiction, which is unavailable where the parties are software components of uncertain domicile. Brand requires an audience with memory and attention, neither of which an agent possesses. What remains is reputation: a portable, cumulative, and continuously updated summary of past conduct that a counterparty can evaluate in the time available before transacting. This is a claim about selection, not about everything an institution must do. Escrow, bonding, attestation, and automated settlement also operate at machine speed, and the six requirements assume they operate alongside the reputation layer. What none of them supplies is the cumulative record of conduct on which selection among strangers can clear.

The second half is where most systems fail. Reputation that is merely recorded is a score. Reputation that constrains behavior requires that participants have something at stake, which is the function of reputation staking and, more generally, of skin in the game (Calcaterra, Kaal, and Andrei 2018; Calcaterra, Kaal, and Sivalingam 2018). A participant who can abandon an identity and acquire a new one at negligible cost has no reputation in the operative sense, however elaborate the scoring.

Three design consequences follow, and each is a matter of engineering rather than of good intentions.

The first concerns identity cost. Reputation is meaningful only where identity is expensive to replace. This does not require legal identity, and the confusion of the two has retarded the field considerably. It requires only that the accumulated standing be more valuable than the cost of starting again, which is a condition a system can create rather than inherit.

The second concerns context. Reputation earned in one domain must not transfer silently to another. Domain specificity was the central design commitment of the earliest architecture in this line (Calcaterra, Kaal, and Andrei 2018) and remains the property most often discarded in implementation, because a single number is easier to display than a vector.

The third concerns decay and contestation. Standing that never decays rewards

incumbency rather than conduct, and standing that cannot be challenged is an assertion. The evolution of decentralization depends on reputation systems that do both (Calcaterra and Kaal 2021), and the extension of these mechanisms to autonomous agent populations has been developed as a governance architecture in its own right (Kaal 2024b; Kaal 2026e).

None of this is novel to the sovereign local runtime. What is novel is the setting. In a platform system, the operator can compensate for a weak reputation layer through unilateral intervention: delisting, refunding, or overriding. In a sovereign runtime there is no operator positioned to intervene. The reputation layer is therefore not a convenience feature. It is the institution of last resort, and it must be built to bear that weight from the beginning.

X. The Institutional Interface: Proof Without Surrender

Enterprises and regulators require verification. Users require non-disclosure. The two requirements are compatible, and the belief that they are not has become a substantial barrier to adoption.

The apparent conflict rests on a conflation of proof with production. An enterprise deploying a sovereign runtime must be able to establish that policy was enforced, that data crossed only permitted boundaries, and that outcomes are attributable. It does not need the data itself in order to establish any of those things. What it needs is a durable artifact, produced at execution time, that a third party can evaluate without access to the underlying content.

Where such artifacts exist, the regulatory posture available to the system changes in kind. The regulator's problem in fast-moving domains is that rules written against a technology are obsolete before they are enforced, a difficulty developed at length in the dynamic regulation literature and its treatment of the pacing problem (Fenwick, Kaal, and Vermeulen 2017; Kaal 2013a; Kaal 2014; Kaal 2017). A system that can demonstrate compliance with stated criteria, rather than conformity to a frozen specification, permits the criteria to be revised as conditions change without reopening the technology.

The longer project of rendering legal requirements machine-interpretable so that such demonstrations can be automated is under development elsewhere (Furrer and Kaal 2025; Furrer, Kaal, and Meyer 2025; Kaal 2025c), as is the analysis of how artificial intelligence is reshaping legal practice itself (Kaal and Gray 2025). The requirement stated here is narrower and prior to all of it. Before legal requirements can be evaluated automatically against a runtime's behavior, the runtime must produce a record of that behavior which survives the session and admits third-party interpretation. Requirement 2 is therefore a precondition for the regulatory interface, not merely for debugging.

XI. Research Agenda and Conclusion

The requirements are testable, and the field should test them.

Three programs follow directly. The first is measurement of the coordination cost

identified in Part IV. The claim that bilateral settlement of the five questions scales super-linearly is stated here analytically and should be established empirically, by instrumenting integration effort across runtimes with and without a common institutional layer. The second is adversarial evaluation of reputation layers against the threat model named in Requirement 4, conducted as registered analysis rather than as demonstration, following the design used for controlled multi-model cohorts (Kaal 2026d). The third is the construction of a portable provenance artifact adequate to Requirement 2. Contributions 1 and 2 in Part VIII specify the minimal form such an artifact would take in one existing runtime, and implementing them, together with the coverage and the periodic root they name, would convert that research program from a design question into a measurement.

A note on method closes the argument. This Article states requirements before remedies, and confines its remedies to minimal modifications of one existing system rather than offering an architecture of its own. The reputation computation, the evaluation methodology, the value-attribution algorithm, and the contestation design that Requirements 4 through 6 would need are developed in the author's separately published work and are deliberately not specified here (Kaal 2026e; Kaal 2026g). That ordering reflects a judgment about sequence rather than a limit on the analysis. The institutional deficit documented across decentralized organizations arose in substantial part because implementations preceded requirements, and governance was retrofitted onto systems whose incentive structures had already hardened (Kaal 2026f; Kaal 2020). The sovereign local agent runtime is early enough that the sequence can still be reversed. Requirements stated now, against systems still under construction, remain testable. Requirements stated later become postmortems.

Read together, the findings say something narrower and harder than that one system lacks fields. Mosaic made the correct foundational choice, placing enforcement in a trusted core and treating executing tools as low-trust, and its public implementation does not yet turn that enforcement into end-to-end institutional evidence: coverage is partial, authority does not travel with the work, and the records that exist cannot demonstrate their own completeness to anyone outside the machine. Local custody is the necessary foundation for accountable agent coordination, not a substitute for it.

Sovereignty returned custody to the user. It did not return accountability to anyone. Whoever writes the coordination rules writes the institution.

Bibliography

HyperCycle Development. 2026. *mosaic-companion* (Mosaic Browser). Version at commit 2d920ceff367ff7a2c73cbf04333572fca465182, pushed July 31, 2026; retrieved August 18, 2026. Apache License 2.0. GitHub. https://github.com/hypercycle-development/mosaic-companion. Files cited: docs/architecture/overview.md, gatekeeper.md, permissions.md, manifest.md, data-model.md, implementation-status.md; electron/integrations/sandbox/chronicle.ts, gatekeeper.ts, types.ts, index.ts, tool-bridge.ts, wasm-launcher.ts; electron/integrations/tools/types.ts; plugins/payments-jit/README.md.

Calcaterra, Craig, and Wulf A. Kaal. 2021. "The Importance of Reputation for the Evolution of Decentralization." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3782210.

Calcaterra, Craig, Wulf A. Kaal, and Vlad Andrei. 2018. "Blockchain Infrastructure for Measuring Domain Specific Reputation in Autonomous Decentralized and Anonymous Systems." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3125822.

Calcaterra, Craig, Wulf A. Kaal, and Gopinath Sivalingam. 2018. "Reputation Protocol for the Internet of Trust: Conceptual Whitepaper." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3266953.

Coase, R. H. 1937. "The Nature of the Firm." *Economica* 4 (16): 386-405.

Fenwick, Mark, Wulf A. Kaal, and Erik P. M. Vermeulen. 2017. "Regulation Tomorrow: What Happens When Technology Is Faster Than the Law." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2834531.

Fenwick, Mark, Wulf A. Kaal, and Erik P. M. Vermeulen. 2018. "Why 'Blockchain' Will Disrupt Corporate Organizations." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3227933.

Furrer, Andreas, and Wulf A. Kaal. 2025. "Universal Digital Law Codex (UDLC): Building the Legal Infrastructure for the Digital Era." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5554218.

Furrer, Andreas, Wulf A. Kaal, and Stephan D. Meyer. 2025. "Universal Digital Law Codex (UDLC)." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5886342.

Kaal, Wulf A. 2013a. "Dynamic Regulation of the Financial Services Industry." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2273857.

Kaal, Wulf A. 2013b. "Evolution of Law: Dynamic Regulation in a New Institutional Economics Framework." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2267560.

Kaal, Wulf A. 2014. "Dynamic Regulation via Governmental Contracts." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2517677.

Kaal, Wulf A. 2016. "Dynamic Regulation for Innovation." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2831040.

Kaal, Wulf A. 2017. "Dynamic Regulation via Contingent Capital." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2957645.

Kaal, Wulf A. 2018a. "Crypto Economics: The Top 100 Token Models Compared." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3249860.

Kaal, Wulf A. 2018b. "Decentralized Mechanical Turk Through Verified Reputation." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3128900.

Kaal, Wulf A. 2019b. "Blockchain Solutions for Agency Problems in Corporate Governance."

SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3373393.
Kaal, Wulf A. 2019c. "Decentralized Commerce: A Primer on Why Decentralized Reputation Verification Systems Are Needed." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3405401.
Kaal, Wulf A. 2020. "Decentralized Autonomous Organizations: Internal Governance and External Legal Design." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3652481.
Kaal, Wulf A. 2021b. "Reputation as Capital: How DAOs Upgrade Finance." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3949098.
Kaal, Wulf A. 2021c. "Reputation as Capital: How Decentralized Autonomous Organizations Address Shortcomings in the Venture Capital Market." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3962614.
Kaal, Wulf A., and Josh Bykowski. 2023. "Decentralized Autonomous Organizations (DAO): A Market Meta Analysis." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4529715.
Kaal, Wulf A. 2024b. "AI Governance via Web3 Reputation System." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4941807.
Kaal, Wulf A. 2024d. "DAO Market Meta Analysis 2024." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5254152.
Kaal, Wulf A. 2025c. "The UDLC DAO: Operationalizing a Continuously Evolving Universal Digital Law Codex." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5887242.
Kaal, Wulf A. 2026a. "AI's Mother's Instinct: Engineered Consequence, Emergent Ethics and the Institutional Trajectory Toward Agentic Alignment." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6244278.
Kaal, Wulf A. 2026b. "The Collapse of Scarcity Economics." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6421319.
Kaal, Wulf A. 2026c. "Computative Economics: A Framework for Economic Analysis under Computational Abundance." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6607458.
Kaal, Wulf A. 2026d. "Empirical Evaluation of the Agentic Reputation Substrate: Deliberation, the Composition of Error, and the Registered Measurement of Agency Costs in a Controlled Multi-Model Cohort." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=7261018.
Kaal, Wulf A. 2026e. "Governance as a Product (GaaP): A Reputation-Weighted Institutional Architecture for Autonomous AI Agent Governance." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6886078.
Kaal, Wulf A. 2026f. "The Institutional Deficit in Decentralized Autonomous Organizations: An Empirical Analysis." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6819121.
Kaal, Wulf A. 2026g. "Architecture of the Agentic Reputation Substrate." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=7260278.
Kaal, Wulf A. 2026h. "Computative Economics: An Empirically Grounded Framework for Economic Analysis under Computational Abundance." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=7261481.
Kaal, Wulf A. 2026i. "Possibility Loops: An Operational Architecture for Computative Economics in Agent Coordination Systems." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6655138.

Kaal, Wulf A., and Morgan A. Gray. 2025. "The Evolving Role of Artificial Intelligence in Law." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5541658.

Kaal, Wulf A., and Hayley Howe. 2021. "Custody of Digital Assets." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3936876.

Kaal, Wulf A., and Erik P. M. Vermeulen. 2016. "Venture Capital as Dynamic Regulation of Disruptive Innovation." SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2740477.

North, Douglass C. 1990. *Institutions, Institutional Change and Economic Performance*. Cambridge: Cambridge University Press.

Williamson, Oliver E. 1985. *The Economic Institutions of Capitalism*. New York: Free Press.